

(19) 日本国特許庁 (J P)

(12) 公表特許公報 (A)

(11) 特許出願公表番号

特表平8-512438

(43) 公表日 平成8年(1996)12月24日

(51) Int.Cl. ⁹	識別記号	庁内整理番号	F I
H 0 3 M 7/30		9382-5K	H 0 3 M 7/30 Z
G 0 9 C 1/00	6 1 0	7259-5J	G 0 9 C 1/00 6 1 0 D
H 0 4 B 1/66		4101-5J	H 0 4 B 1/66
H 0 4 L 9/18		8842-5J	H 0 4 L 9/00 6 5 1

審査請求 未請求 予備審査請求 有 (全 93 頁)

(21) 出願番号 特願平6-520072
 (86) (22) 出願日 平成6年(1994)2月28日
 (85) 翻訳文提出日 平成7年(1995)9月8日
 (86) 国際出願番号 PCT/US94/02088
 (87) 国際公開番号 WO94/21055
 (87) 国際公開日 平成6年(1994)9月15日
 (31) 優先権主張番号 08/030,741
 (32) 優先日 1993年3月12日
 (33) 優先権主張国 米国 (US)
 (81) 指定国 EP(AT, BE, CH, DE, DK, ES, FR, GB, GR, IE, IT, LU, M C, NL, PT, SE), AU, CA, JP

(71) 出願人 ザ ジェイムス グループ、インク
 アメリカ合衆国、フロリダ州 33942、ナ
 プレス、ノース エアポート ロード
 125、スート 202
 (72) 発明者 ジェイムス、ディビッド、シー
 アメリカ合衆国、フロリダ州 33937、マ
 ルコ アイランド、バルド イーグル ド
 ライブ 1085、アパートメント F-608
 (74) 代理人 弁理士 木村 満 (外2名)

(54) 【発明の名称】 データ圧縮方法

(57) 【要約】

高度にランダム化されたデータを圧縮する方法を含むデータ圧縮方法が開示されている。あまりランダム化されていないデータを圧縮するためにニブル符号化、分配符号化、直接ビット符号化法が開示されている。さらに、高度にランダム化されたデータの圧縮に非常に有効なランダムデータ圧縮ルーチンも開示されている。開示された圧縮方法の全ては、ビットレベルで動作する。従って、圧縮対象データの性質や生成原因には無関係である。従って、この発明の方法は、その生成源によらず、あらゆる形式のデータに広く適用可能である。

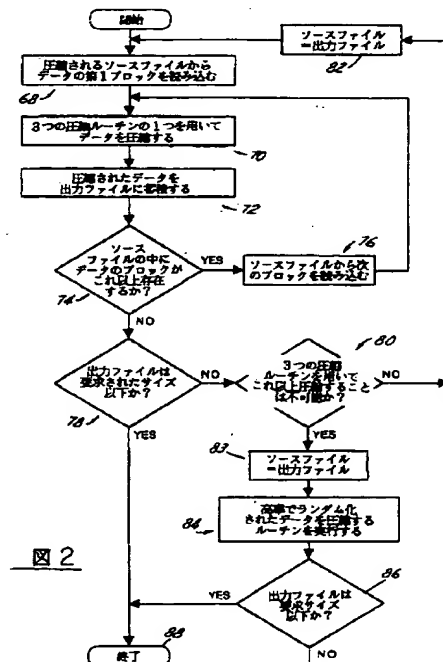


図 2

09/727,096

【特許請求の範囲】

1. 以下のステップを有し、電氣的にコード化された一連のバイナリデータのストリームから冗長抽出方法、
 - A) 前記一連のデータから n ビットを分析し、
 - B) n ビットの複数の全ての可能な状態に基づいて前記解析した n ビットの状態を決定し、
 - C) Bステップの結果に基づいて、前記分析した n ビットの状態を複数のコードの1つに関連付け、
 - D) ステップCのコードを第1と第2の部分に分解し、
 - E) 前記第1の部分を第1と第3の位置の少なくとも1つに割り付け、
 - F) 前記第2の部分を少なくとも第2の位置に割り付ける。
2. 分析している2ビットを含む n ビットをパス（通過）することを特徴とする請求項1に記載の冗長抽出方法。
3. ステップFの割り付けは、さらに、前記第1の部分を前記第1と第3の位置の少なくとも1つの内容に連結することを含むことを特徴とする請求項1に記載の冗長抽出方法。
4. ステップFの割り付けは、さらに、前記第2の部分を少なくとも前記第2の位置の内容に連結することを含むことを特徴とする請求項1に記載の冗長抽出方法。
5. D) 前記第1、第2及び第3の位置の少なくとも1つの内容を前記一連のデータに割り付け、前記A乃至Fを繰り返すステップをさらに含むことを特徴とする請求項1に記載の冗長抽出方法。
6. 以下のステップを有し、一連のバイナリデータを構成する方法、
 - A) 前記一連のデータから n ビットを分析し、
 - B) n ビットの複数の可能な状態に基づいて前記解析した n ビットの状態を決定し、
 - C) Bステップの結果に基づいて、第1、第2、第3の位置の少なくとも1つにおける前記 n ビットの状態をコード化し、ここで、前記状態のコード化は複数

の可変長コードの生成と、前記複数のコードの少なくとも一部を少なくとも第1、第2及び第3の部分に分割し、前記コード又は前記コード部分の少なくとも1つを前記第1、第2及び第3の位置の少なくとも1つに、少なくとも一部において前記分析した n -ビットの所望の属性の関数として、割り付ける。

7. 前記 n は2であることを特徴とする請求項6に記載の一連のバイナリデータを構成する方法。

8. 前記所望の特徴は前記 n -ビットの値の集団を含むことを特徴とする請求項1に記載の一連のバイナリデータを構成する方法。

9. 前記複数のコードからの第1コードを前記分析した n -ビットの所望の属性に関連付けるステップをさらに含む、前記第1のコードは $n-1$ ビット長以下であることを特徴とする請求項6に記載の一連のバイナリデータを構成する方法。

10. D) 前記第1、第2及び第3の位置の少なくとも1つの内容を前記一連のバイナリデータに割り付け、前記A乃至Fを繰り返すステップをさらに含むことを特徴とする請求項6に記載の一連のバイナリデータを構成する方法。

11. 以下のステップを有し、電氣的にコード化された一連のバイナリデータのストリームから冗長データを抽出して圧縮する方法、

A) 前記一連のデータから n -ビットを分析し、

B) n -ビットの複数の可能な状態に基づいて前記解析した n -ビットの状態を決定し、

C) コードをBステップで決定された状態に関連付け、

D) ステップCのコードの第1の部分を実第1と第3のレジスタのうちの少なくとも1つの内容に連結し、

E) 前記コードの第2の部分を実第2のレジスタの内容に連結し、

F) 前記第1、第2、第3のレジスタの少なくとも1つの内容から一度に n -ビットを分析し、

G) ステップFで分析した n -ビットのそれぞれにコードを割付け、前記コードの値は前記ステップFで分析したビットパターンに一部分で依存し、

H) 前記コードを複数の部分に分割し、前記部分を前記第1、第2、第3の

レジスタの少なくとも2つに割り付ける。

12. 以下のステップを有し、電氣的にコード化された一連のバイナリデータのストリームから冗長データを繰り返して抽出して圧縮する方法、

- A) 前記一連のデータからnビットを解析し、
- B) 望ましいnビットパターンの発生と、望ましくないnビットパターンの発生を分類し、
- C) 望ましいビットパターンの各発生について、第1のコード列を第1と第2のレジスタの内容に関連付け、
- D) 望ましくないビットパターンの各発生について、第2のコード列を第2のレジスタの内容に関連付け、
- E) 第1レジスタの内容を圧縮し、
- F) ステップEの圧縮された内容の少なくとも一部を一連のバイナリデータに割り付け、
- G) A乃至Fのステップを繰り返す。

13. 以下のステップを有し、電子的装置の使用によりデータを圧縮する方法、

- A) データ源からデータを読み出し、
 - B) 前記データを使用可能なフォーマットに変換し、
 - C) 前記データを暗号化して、前記データ源から読み出したデータの総量減
- 量し、

- D) 暗号化されたデータを保存し、
- E) 保存されたデータを読み出し、前記変換、暗号化、及び保存ステップを保存データが所望レベルに減少するまで繰り返す。

14. 前記情報の変換は、ビットプリドミナンス法を使用して達成することを特徴とする請求項13に記載の方法。

15. 前記暗号化は直接ビット操作を含むことを特徴とする請求項13に記載の方法。

16. 前記暗号化はニブル暗号化を含むことを特徴とする請求項13に記載の方法。

17. 前記暗号化は分配法を含むことを特徴とする請求項13に記載の方法。

18. 前記分配はパックルーチンを含むことを特徴とする請求項17に記載の方法。

19. 前記情報は予め圧縮されたデータであることを特徴とする請求項13に記載の方法。

20. 以下のステップを有し、ニブル暗号技術を用いてバイナリデータ列から冗長を抽出する方法、

A) バイナリデータ列からニブルを分析し、

B) ステップAにおいて分析されたニブルの値に応答し、複数の制御ワードの1つを7つの出力列の少なくとも1つに連結する。

21. 以下のステップを有し、分配圧縮法を用いて電氣的にコード化されたバイ

ナリデータ列からのデータを圧縮する方法、

A) 前記バイナリデータ列を分析して、最も頻繁に発生するニブル値を決定し、

B) 前記データ列を所定サイズのブロックに分割し、

C) どのブロックが最も頻繁に発生するニブルを含み、どのブロックがそうではないかを示す第1のコード化されたストリングを生成し、

D) 最も頻繁に発生するニブルを含んでいないブロックに関して、ベース256パッキングを用いてその内容を圧縮し、

E) 最も頻繁に発生するニブルを含んでいるブロックに関して、最も頻繁に発生するニブルが起こる位置を各ブロックについて示す第2のコード化されたストリングを生成し、

F) 第1と第2のコード化されたストリングを縮小する。

22. 以下のステップを有し、直接ビット操作法を用いて電氣的にコード化されたバイナリデータ列からのデータを圧縮する方法、

A) 前記バイナリデータ列を複数の入力ワードに分解し、

B) 複数のレンジを定義し、

C) 各ワードが属するレンジを決定し、

D) ステップCで成された各決定について、各ワードをバランス値に変換し、制御ワードと分析ワードを各バランス値に付加し、バランス値と制御ワードと分析ワードはそれらに関連する入力ワードの値を個別的に決定する。

23. 以下のステップを有し、電氣的にコード化され、ランダムに分配したデータ列を縮小する方法、

- A) 前記データ列をランダムに分配したデータの複数のブロックに分解し、
- B) 前記ランダムに分配したデータのブロックの1つを選択し、
- C) ステップBで選択されたブロックを第1と第2の部分に分解し、
- D) ステップBで選択されたブロックの第1の部分内の所定のワードの発生をカウントし、

E) 前記第2の部分のデータを圧縮する。

24. 前記圧縮はパッキングを使用することを含むことを特徴とする請求項23に記載の方法。

25. 前記所定のワードは、値が0の単一のビットセットと値が1の単一のビットセットから構成されるセットから選択される、ことを特徴とする請求項23に記載の方法。

26. 前記第1の部分と前記圧縮された第2の部分とを結合し、結合されたデータに等しいデータ列を設定し、A乃至Eのステップを繰り返す、ことを特徴とする請求項23に記載の方法。

27. 以下のステップを有し、電氣的にコード化された入力データの列から、パッキング及びアンパッキングが容易な電氣的にコード化されたデータのコード化された可変長の列を生成する方法、

- A) 前記入力データ列をnビットずつ分析し、
- B) nビットの複数の全ての可能な状態に基づいて、各分析されるnビットについてステータスワードをコード化し、
- C) 前記コードの少なくともいくつかを第1と第2の部分に分解し、
- D) 前記コードの第1部分を第2の保存レジスタにセットし、
- E) 前記コードの第2部分を第1と第3の保存レジスタの少なくとも1つ

にセットし、

F) 第2のレジスタにおける第1のワード値の発生数が前記第1のレジスタの内容の長さを指示するように、ステータスワードのコーディングに使用する値を選択し、前記第1と第2の部分の配置を選択する。

28. 第2のレジスタにおける第2のワード値の発生数が前記第3のレジスタの内容の長さを指示するように、ステータスワードのコーディングに使用する値を

選択し、前記第1と第2の部分の配置を選択する、ことを特徴とする請求項27に記載の方法。

29. 第1のレジスタにおける第2のワード値の発生数が前記第3のレジスタの内容の長さを指示するように、ステータスワードのコーディングに使用する値を選択し、前記第1と第2の部分の配置を選択する、ことを特徴とする請求項27に記載の方法。

30. $n = 2$ であることを特徴とする請求項27に記載の方法。

31. 前記コードワードは2ビット長であることを特徴とする請求項27に記載の方法。

32. 前記第1のワード値は値が0の単一ビットセットと値が1の単一ビットセットからなるセットの中から選択されることを特徴とする請求項27に記載の方法。

33. 以下の工程を有し、電氣的にコード化された一連のデータを容易にパックすることができる電氣的にコード化されたデータ列に分配する方法、

A) データを少なくとも第1と第2のデータ列に変換し、

B) 前記第1のデータ列を少なくとも第1のコードワードを含むようにコード化し、

C) 前記第1の列中の第1のコードワードの発生数により前記第2のデータ列の長さを示す。

34. 前記データを少なくとも第1、第2、及び第3のデータ列に変換し、前記第1の列を少なくとも第1と第2のコードワードを含むようにコード化し、前記第1の列中の第2のコードワードの発生数により第3のデータ列の長さを示す

、ことを特徴とする請求項 3 3 に記載の方法。

3 5. 前記データを少なくとも第 1、第 2、及び第 3 のデータ列に変換し、前記第 2 のデータセットを第 2 のコードワードを含むようにコード化し、前記第 2 の列中の第 2 のコードワードの発生数により第 3 のデータ列の長さを示す、ことを特徴とする請求項 3 3 に記載の方法。

3 6. 前記第 1 と第 2 の列を連結して格納する、ことを特徴とする請求項 3 3 に記載の方法。

3 7. 前記第 1 と第 2 と第 3 の列を連結して格納する、ことを特徴とする請求項 3 4 に記載の方法。

3 8. 前記第 1 と第 2 と第 3 の列を連結して格納する、ことを特徴とする請求項 3 5 に記載の方法。

【発明の詳細な説明】

データ圧縮方法

技術分野

この発明は一般的にデータ圧縮方法に関し、特に、デジタル型式で表現されたデータを圧縮するためにデジタル処理装置を動作させる方法に関する。

発明の背景

データ圧縮技術は、通信及びコンピュータ分野で広く用いられている。通信分野においては、受信時にデータを元の形に再構成できる圧縮データを送信することが望ましい。圧縮データの送信は、非圧縮の同一データの送信よりも送信時間が短い。コンピュータ分野では、圧縮データは非圧縮データよりも記憶領域面で有利である。したがって、固定記憶容量を持った記憶装置においては、圧縮する方がより多くのファイルを蓄積することができる。よって、データ圧縮による2つの主な利点としては、記憶容量の増大と転送時間の短縮とがある。

データ圧縮技術は損失があるタイプ（損失型）と損失がないタイプ（非損失型）の2つの大きなカテゴリーに分類される。非損失型データ圧縮技術は、圧縮や伸張（圧縮解凍）処理において、情報の損失が許されない時に使用される。損失型データ圧縮技術は非損失型データ圧縮技術よりも正確さに欠けるが、一般的に処理速度が速い。つまり、損失型データ圧縮技術は圧縮や伸張のサイクルでの処理速度のために正確さをやや犠牲にしている。損失型データ圧縮技術は、情報の損失を許容できるアプリケーション（例えば、デジタル化したビデオデータ及びオーディオデータの送信、記憶等）の処理時に典型的に使用される。損失型データ圧縮は非損失型データ圧縮よりも圧縮率が大きく、圧縮処理速度も速い。損失型データ圧縮技術は近時、パーソナルコンピュータとその関連市場用のオーディオ及びビデオのアプリケーションの普及という観点から相当な重要性を得てきている。それ以外の大多数のアプリケーションはデータ圧縮において、非損失型データ

ータ圧縮技術を使用している。

定義によれば、非損失型データ圧縮技術はデータが圧縮や伸張のサイクルを通

過した後、データの正確な復元を保証する方法を使用している。非損失型圧縮はコンピュータで使われているデジタルデータの記憶と最も関連している。そのアプリケーションはデータベースレコード、表計算ソフト、ワープロの記憶を含む。

根本的に、全てのデータ圧縮技術は情報理論として知られる数学分野に関連し、かつこれを用いている。この数学分野は情報の表現、蓄積、通信に関する問題に関係している。

データ圧縮は冗長性との関係より、情報理論のその分野に関連している。データメッセージ中の情報が冗長ならば（その欠落がデータ内にコード化されている情報を減少させないならば）、メッセージ内にコード化されている情報を欠くことなく、データメッセージを短縮することができる。したがって、非損失型データ圧縮は、メッセージが有する情報を破壊することなく完全なまま、データメッセージのサイズを縮小する。

エントロピーは、どのくらいの情報がメッセージ内でコード化されているかを数量的に表すための用語である。高いエントロピーを持つメッセージは、同一の長さで、低いエントロピーを持つメッセージより、情報量が多い。メッセージ（通信データ）中の記号（文字）のエントロピーはそのメッセージ内において任意の文字が発生する確率の負の対数として定義される。複数ビットからなる文字の情報内容を決定するために、以下に示す2を基底とした対数を用いてエントロピーを表す。

$$E_{\text{har}}(X) = -\log_2 (\text{文字}(X) \text{の確率})$$

・ $E_{\text{har}}(X)$ = メッセージ中の任意の文字のエントロピー

・ 文字(X)の確率 = そのメッセージにおいて文字(X)が発生する確率
 メッセージ全体のエントロピーはそのメッセージ中に存在するキャラクタ（又は記号）のエントロピーの単純な合計である。

エントロピーの概念は、データ圧縮技術を最適化するための指示を与える。なぜなら、その概念は、情報のどの程度のビット数がメッセージ中に実際に存在するかを論理的に決定するためである。もし、任意のメッセージ中で 'Q' の文字

が1/16の確率で発生するならば、その文字が有する情報の内容は、4ビットであるということであり、'QQ'は8ビットである。'QQ'を表現するために標準8ビットのASCIIコードを使用すれば、16ビット必要になる。エントロピーの8ビットとデータ列をコード化（エンコード）するために使用される16ビットの違いにより、データ圧縮が可能になる。

情報理論で使用される場合、エントロピーは情報内容の相対的な尺度であり、絶対的な尺度にはなり得ない。なぜなら、キャラクタの情報の内容は任意のメッセージ中でのそのキャラクタの発生率によるためである。2つの違うメッセージにおいて、両方が少なくとも1回の文字'E'の発生回数を持つならば、'E'の発生率は2つのメッセージの間で違う値になるだろう。したがって、文字'E'の情報の内容は固定された値ではなく、文字'E'の発生率に比例してメッセージ毎に変化するものである。ほとんどのデータ圧縮技術は、任意のメッセージ中で高い発生率を持つ記号（又は文字）を予測する。高い発生率を持つ記号は低い発生率を持つ記号と比較して、必然的に低い情報の内容を持ち、コード化するために必要なビットも少ない。他の技術としては、任意の文字の発生率を確定する方法が知られている。文字データにとって、最も容易な方法は各々の文字の発生率を確定し（実験的に）、バイナリコードの長さが文字の発生率に反比例するように、各々の文字にバイナリコードを割り当てることである。（すなわち、最も短いバイナリコードは最も頻繁に現れる文字に割り当てられる。）

辞書を基にした技術は、データの一部又は複数の部分を走査して、どの文字又は文字列が最も頻繁に現れるかを判定する方法を用いている。文字又は文字列は辞書の中に配置されていて、文字又は文字列の発生率に反比例したコード長の所定のコードが割り当てられている。文字又は文字列はデータファイルから読み込まれ、その文字の辞書登録（見出し語）とマッチングされ、対応したコードを用いてコード化される。

近時では、データ圧縮のソフトウェアはDOSの世界において急増している。これらのソフトウェアはいくつかの欠点を有する。第1に、プログラムは典型的なディスク集中型で、その結果、プログラムの動作はディスクに読み書きするスピ

ードに規制されている。例えば、ランダムアクセス速度が18ミリ秒のハードディスクを装備した25MHZ 386 GATEWAY 2000™で作動するPKZIP™として広く知られるコンピュータ圧縮プログラムは1メガビットのASCIIファイルをその1/2のサイズに圧縮するために8.5秒かかる。データベースファイル及び表計算ソフトウェアの圧縮もほぼ同一の時間がかかるが、それらは元の2/3のサイズにしか圧縮されない。バイナリファイルが最も圧縮され、元のサイズの10~40%となる。しかし、同一の長さのASCIIファイルよりも圧縮時間が6倍かかる。

以上のデータ圧縮方法の欠点を考慮すると、より効果的及び効率的なデータ圧縮技術が必要となる。

この発明は、これまで達成されなかったデータ圧縮のレベルに達したと考えられる。既知のデータ圧縮製品はテキストファイル及びグラフィックファイルに対して50%以上の圧縮率は不可能であり、実行ファイルに対しては更に低い(約45%圧縮)。この発明を用いれば、90%のデータ圧縮レベル(特定なアプリケーションを使用すればより高い圧縮率で)が、現在のデータ圧縮製品が同一のデータを50%の圧縮率で圧縮するのと同じか又はそれ以下の時間で達成される。この発明は、ランダム(又はカオス)に出現する形式の情報から、順序付けられた情報列を分離したり、配置したりすることによって高い圧縮率を達成する。従来のデ

ータ圧縮方法は、ランダムに(冗長なしに)配置されているように見えるデータ中の秩序(冗長)を見出できない場合が多かった。結果的に、従来の圧縮方法は、検出できない秩序を圧縮することにおいて非効率的である。

以上の説明から理解できるように、順序付けられたデータがランダムデータから抽出されれば、そのように順序付けられたデータは容易に圧縮できる。

この発明の第2の面は高い圧縮率を達成する性能を更に高めるものであり、再帰的に(繰り返して反復的に)その性質をデータに適用できることである。特に、この発明の方法は、ファイルを圧縮する度に、複数のパスを作成していくことが可能である。このようにして、一連の反復処理は所望の圧縮レベルが達成されるまで、実行される。

発明の概要

第1の観点において、この発明はバイナリデータ列から冗長を抽出する方法を提供する。この方法は、前記一連のデータから所定数のビットを分析（解析）し、分析したビットの複数の可能な状態に基づいて前記分析したビットの状態を決定することを含む。分析したビットの状態に基づいて、分析したビットの状態に状態コードが関連付けられる。状態コードは第1と第2の部分に分解される。ここで、第1の部分は第1と第3の位置の少なくとも1つに割り付けられ、前記第2の部分は少なくとも第2の位置に割り付けられる。

第2の観点において、この発明はバイナリデータ列を圧縮する方法を提供する。この方法は、前記一連のバイナリデータから所定数のビットを分析（解析）するステップを含む。各分析したビットの組について、ビットの状態が決定され、状態コードが、第1、第2、第3の位置の少なくとも1つに設定される。状態のコード化（コーディング）は、複数の可変長コードの生成と、前記複数のコードの少なくとも一部を少なくとも第1と第2の部分に分割することを含む。前記コ
ー

ド又は前記コード部分の少なくとも1つを、前記第1、第2及び第3の位置の少なくとも1つに、少なくとも一部において前記分析した n -ビットの所望の属性の関数として、割り付ける。

第3の観点において、この発明は、一連のバイナリデータから冗長データを繰り返し抽出して圧縮する方法を提供する。この方法は、前記一連のデータから n ビットを分析（解析）し、所望の n ビットパターンと非所望の n ビットパターンの発生を分類することを含む。第1のコード列は、所望のビットパターンの各発生により、第1と第2のレジスタの内容に連結され、第2のコードシーケンスは、非所望のビットパターンの各発生により、第2と第3のレジスタの内容に連結する。第1のレジスタの内容は圧縮され、圧縮された内容の少なくとも一部は、再び、同一の方法を用いて処理される。

さらに、第4の観点において、この発明は、一連のバイナリデータから冗長データを抽出して圧縮する方法を提供する。この方法は、前記一連のデータから所定数のビットを分析（解析）し、複数の可能な状態に基づいて前記解析したビッ

トの状態を決定する。状態にコードが関連付けられ、このコードは複数の部分に分割される。コードの第1の部分は第1と第3のレジスタの少なくとも一方の内容に連結され、コードの第2の部分は、第2のレジスタの内容に連結される。第1～第3のレジスタの内容。前記第1、第2、第3のレジスタの少なくとも1つが選択され、その内容はnビット単位で分析される。レジスタの1つの分析されたnビットのそれぞれにコードが割り当てられる。コード値は、前記レジスタの内容から分析されたビットのビットパターンに基づいている。コードは第1～第3のレジスタの少なくとも2つの部分に分割されている。

さらに他の観点において、この発明は、デジタルコンピュータを用いてデータを圧縮する方法を開示する。この方法は、データ源からデータを読み出し、前記データを使用可能なフォーマットに変換し、前記データを暗号化して、前記データ源から読み出したデータの総量を減量し、暗号化されたデータを保存し、保

存されたデータを読み出し、前記変換、暗号化、及び保存ステップを保存データが所望レベルに減少するまで繰り返す。

さらに、この発明は、ニブル暗号化技術を用いてバイナリデータ列から冗長を抽出する方法を開示する。この技術は、バイナリデータ列からニブルを分析し、分析されたニブルの値に応答し、複数の制御ワードの1つを7つの出力列の少なくとも1つに連結する。

さらに、第7の観点において、この発明は、分配圧縮法を用いてバイナリデータ列からのデータを圧縮する方法を提供する。この方法は、前記バイナリデータ列を分析して、最も頻繁に発生するニブル値を決定し、前記データ列を所定サイズのブロックに分割し、どのブロックが最も頻繁に発生するニブルを含み、どのブロックがそうではないかを示す第1のコード化されたストリングを生成し、最も頻繁に発生するニブルを含んでいないブロックに関して、ベース256パッキングを用いてその内容を圧縮し、最も頻繁に発生するニブルを含んでいないブロックに関して、最も頻繁に発生するニブルが起こる位置を各ブロックについて示す第2のコード化されたストリングを生成し、第1と第2のコード化された列を縮小する。

第 8 の観点において、この発明は、直接ビット操作法を用いてバイナリデータ列からのデータを圧縮する方法を開示する。この方法は、バイナリデータ列を複数の入力ワードに分解し、複数のレンジを定義し、各ワードが属するレンジを決定し、各ワードをバランス値に変換し、制御ワードと分析ワードを各バランス値に割付け、バランス値と制御ワードと分析ワードはそれらに関連する入力ワードの値を個別的に決定する。

第 9 の観点において、この発明はランダムに分配したデータの列を圧縮する方法を提供する。この観点は、前記データ列をランダムに分配したデータの複数のブロックに分解するステップと、前記ランダムに分配したデータのブロックの 1

つを選択するステップと、選択されたブロックを第 1 と第 2 の部分に分解するステップと、前記ブロックの第 1 の部分内の所定のワードの発生数をカウントするステップと、前記第 2 の部分のデータを圧縮するステップを含む。

第 10 の観点において、この発明は、入力データの列から、パック化及びアンパック化が容易なコード化された可変長の列を生成する方法を開示する。この方法は、前記入力データ列を n ビットずつ分析するステップと、各分析される n ビットについてステータスをコード化（コーディング）するステップを備える。前記コードの少なくとも幾つかは第 1 と第 2 の部分に分解され、前記コードの第 1 の部分の少なくとも幾つかが第 2 の保存レジスタにセットされ、前記コードの第 2 の部分の少なくとも幾つかが第 1 と第 3 の保存レジスタの少なくとも 1 つにセットされる。コードワード値とコード化されたワードの部分の配置は、第 2 の記憶レジスタにおける第 1 のワード値の発生数が前記第 1 のレジスタの内容の長さを指示するように、選択されている。

第 11 の観点において、この発明は、データを容易にパックすることができるデータ列に配置する方法を開示する。この方法は、データを少なくとも第 1 と第 2 のデータ列に変換するステップと、前記第 1 のデータ列を少なくとも第 1 のコードワードを含むようにコード化するステップを含む。第 2 のデータ列の長さは、前記第 1 の列中の第 1 のコードワードの発生の数により示される。

この発明の他の効果とメリットは以下の実施例の説明、請求の範囲の記載、及

び図面を参考にすることにより理解できる。以下に図面を簡単に説明する。

図面の簡単な説明

図1は、デジタルデータ源のダイアグラム図である。

図2は、この発明の圧縮方法を示す一般的なフローダイアグラムである。

図3Aは、この発明のニブル圧縮プログラムを示す一般的なフローダイアグラムである。

図3Bは、この発明の分配圧縮プログラムを示す一般的なフローダイアグラムである。

図3Cは、この発明の直接ビット圧縮プログラムを示す一般的なフローダイアグラムである。

図4は、この発明のニブルコード化法を示す表である。

図5Aと5Bは、所定の入力列に対して図4の表を適用した例を示す。

図6は、この発明のニブルコード化法により処理されたデータをデコードするための復元木を示すフローダイアグラムである。

図7Aは図4のニブルコード化法を用いた入力列のコード化を示す表である。

図7B～7Eは、図6の復元木を用いて図7Aの入力列をデコードすることを示す表である。

図8は、ニブルコード化法によりコード化された列をパッケージするための列パッケージシーケンスを示す。

図9は、図8に示される列パッケージシーケンスをデコードするためのフローダイアグラムである。

図10は、この発明の変換／暗号化ルーチンのフローダイアグラムである。

図11は、表2の0が優位なアルゴリズムを用いてコード化されたデータをデコードするための復元木を示す。

図12は、表3の1が優位なアルゴリズムを用いてコード化されたデータをデコードするための復元木を示す。

図13Aは、表2の変換法を用いて入力列を変換する応用を示す表である。

図13B-13Gは、図11の復元木を用いて図13Aの入力列のデコードを示す表である。

図14は、この発明の変換／暗号化アルゴリズムを示すジェネラルフローチャートである。

図15は、表4の暗号アルゴリズムによりコード化された情報をデコードする復元木を示す。

図16Aは、表4の暗号アルゴリズムを用いた入力列の暗号化を示す表である。

図16B-16Fは、図15の復元木を用いた入力列の暗号化を示す表である。

図17A-17Bは、所定の入力列への0優位ルーチンの適用を示す表を示す。

図18は、単一の入力列に0優位ルーチンを適用することにより3つの出力列を生成する様子を示す図である。

図19は、列TC1\$に図14の変換／暗号化アルゴリズムを適用する様子を示す図である。

図20は、図19のグラフィカル方法を表す表形式の図である。

図21は、列TC2\$に変換／暗号化アルゴリズムを適用する様子を示す図である。

図22は、列TE212\$に変換／暗号化アルゴリズムを適用する様子を示す図である。

図23は、列TE312\$に変換／暗号化アルゴリズムを適用する様子を示す図である。

図24は、列TC3\$に変換／暗号化アルゴリズムを適用する様子を示す図である。

図25は、所定の入力列に表4の暗号化アルゴリズムを適用する様子を示す表形式の図である。

図26は、列TC1\$に変換／暗号化アルゴリズムを適用する様子を示す表形

式の図である。

図27Aは、任意の入力列に適用する変換／暗号化アルゴリズムの適用を示す図である。

図27Bと27Cは、図27Aの入力列に変換／暗号化アルゴリズムの適用を示す表形式の図である。

図28は、図27Aに示される変換／暗号化アルゴリズムにより生成された列をパックするために使用する列パッケージシーケンスである。

図29は、高度にランダム化されたデータを圧縮する方法を示すダイアグラムである。

図30Aと30Bは、図29の方法で使用するルックアップテーブルの例を示す。

図31は、図29のルーチンで圧縮されたデータを伸張（解凍）するルーチンのフローダイアグラムである。

図32は、この発明の分配コード化法のフローダイアグラムである。

図33Aと33Bは、図32の方法を所定の入力列に適用した例を示す表である。

図34は、図32の方法で生成された列をパッケージングするために使用される列パッケージシーケンスを示す。

図35は、この発明の直接ビットコード化法を示す表である。

図36は、5つの所定のビット値に適用したときの図35の直接ビットコード化法を示す。

図37は、直接ビット操作法によりコード化されたデータをデコードするための復元木を示す。

図38A-38Cは、直接ビット操作方法により繰り返して処理されるデータ列を構築する方法を示す。

図39は、2バイト長のワード値に適用したときの直接ビットコード化法を示す第2実施例である。

好適実施例の詳細な説明

次に示す定義はこの発明を説明する目的のために提示されている。

定義

” バイアス ”

バイナリデータの集合の中の0の数と比較される、同一のデータの集合の中の1の数の度合い。その度合いは比率として表現されている。例えば、任意のバイナリデータの集合が、1つの1に対して1つの0を含むならば、それは均等にバイアスされているという（すなわち50%のバイアス）。列が1より0を多く含んでいれば、それは0にバイアスされている。

” ビジット ”

2進法の桁。例えば1バイトは8ビジットの長さを持つ2進数である。ビジットは10進法のデジットに相当する。

” ビット ”

Binary digIT の短縮形。0か又は1によって表現される2進法の桁。ビジットの短縮形。

” バイト ”

8桁の2進数。

” エントロピー ”

システムにおける秩序の尺度。情報理論の分野では、任意のメッセージに含まれた情報量の尺度。”絶対的なエントロピー”の概念は表現し難いままである。メッセージ中に特定の記号（又は記号の群）の発生率が高ければ高いほど、その記号のエントロピーは少なくなる。

” 文字のエントロピー ”

文字のエントロピーはその文字の発生率の負の対数として定義されている。2進数の文字の情報内容を決定するために、2を基底とした対数を用いてそのエントロピーを表現する。

$$\text{ビットの数} = -\log_2 (\text{発生率})$$

メッセージ全体のエントロピーはそのメッセージ内の個々の文字のエントロピー

一の単純な合計である。例えば、この明細書に発生する文字 "e" の確率が $1/16$ ならば、その文字が有する情報の内容は 4 ビットである。従って、"eeeeee" という文字列が発生すれば、その文字列は合計 20 ビットの情報内容を持つ（各文字の確率を全ての文字に対して合計する）。

” エントロピーの限度 ”

任意のデータの集まりが情報の損失なしには縮小することができない、理論上最小限のサイズ。

” 均等に分配されたデータ ”

均等に分配されたデータはランダムに分配されたデータと同意語である。ランダムに分配されたデータにおいては、文字セット中の各文字は、その文字セット中の他の文字と同じ位メッセージ中に頻繁に現れる。

” ${}_nC_r$ ”

n 個の要素から 1 度に r 個取る場合の組合せの数を指定する数学上の表現。数学上では、 ${}_nC_r$ は次と同等である。

$${}_nC_r = n! / r!(n-r)!$$

・ n, r = 一度に r 個取られる事象の数

・ $!$ = 階乗演算子

例：8 ビットの内の 3 ビットが 1 で 5 ビットが 0 の場合、いくつの違った 8 ビットのパターンを発生させられるか。

答え： ${}_8C_3 = 8! / 3!(8-3)! = 56$

” ニブル ”

4 桁の 2 進数。

” パックルーチン ”

アイテム又は数値を蓄積するために必要なスペースを小さくするために、2 つ又はそれ以上のアイテム（又は数値）を 1 つの単語に結合するソフトウェアルーチン。よく知られているルーチンとしては、PACKED DECIMAL ルーチンがある。パックルーチンの一例を以下に示す。

例：パックルーチン使用の際は、” 基本値 ” が定義されなければならない。基

本値は、パックされる最も大きい発生値よりも大きい値と常に等しい。例えば、次の3つのニブルをパックしたいとする。

$$N1 = 1011$$

$$N2 = 0001$$

$$N3 = 0111$$

12は発生している最も大きいニブル値よりも大きいため、12の基本値が設定されなければならない。

$$\text{基本値} = 11_{10} + 1_{10} = 12_{10} \Rightarrow (1100)_2$$

$$\begin{aligned} \text{パックされた } N1, N2, N3 &= N1 * (\text{基本値})^2 + N2 * (\text{基本値}) + N3 \\ &= (1011)(1100)^2 + (0001)(1100) + (0111) \\ &= (11000110000) + (1100) + (0111) \\ &= (11001000011) \end{aligned}$$

$$\text{パックされた時の長さ}(N1+N2+N3) = 11\text{ビット}$$

$$\text{パックされない時の長さ}(N1+N2+N3) = 12\text{ビット}$$

$$\text{節約された長さ} = 12 - 11\text{ビット}$$

$$= 1\text{ビット}$$

$N1, N2, N3$ は以下に示す方法を用いてアンパックされる。

$$\begin{aligned} \text{アンパック } N1 &= \text{INT} (\text{パックされた } N1, N2, N3) / (\text{基本値})^2 \\ &= \text{INT} (11001000011) / (1100)^2 \\ &= \text{INT} [1011.00100001] \\ &= 1011 \end{aligned}$$

$$\begin{aligned} \text{アンパック } N2 &= \text{INT} (\text{パックされた } N1, N2, N3) - (N1)(\text{基本値})^2 / (\text{基本値}) \\ &= \text{INT} [(11001000011) - 11000110000] / (1100) \\ &= \text{INT} [0001.100101] \\ &= 0001 \end{aligned}$$

$$\begin{aligned} \text{アンパック } N3 &= [(\text{パックされた } N1, N2, N3) - (N1)(\text{基本値})^2] - [(N2)(\text{基本値})] \\ &= [(11001000011) - (11000110000)] - [(0001)(1100)] \\ &= 0111 \end{aligned}$$

ここで図1について言及すれば、この発明の方法は、全ての形式のデジタル型にコード化（エンコード）された情報を圧縮するのに適する。開示された方法はバイナリデータのパターンに焦点を当てることにより動作するので、それらはデータのソースの特性に無関係である。従って、この発明の方法は、デジタル記録媒体に蓄積されるデータファイル50からのデータ、音声情報52を合成するものとして作成されている、又は作成されるであろう信号、デジタル型で蓄積されている他の形式の情報54、マルチメディア56又はグラフィックファイル58にのそれぞれの良好に動作する。更に、この発明の方法は、あらゆるタイプのデジタル記憶装置60からの情報、又はモデム経由で送信されるデジタル

情報62をもうまく処理する。この発明の方法は、Macintosh[®] 64、PC互換機66又は、他のデジタル処理装置等のあらゆる形式のデジタル型処理装置上で動作するのに適している。この発明の方法は専用の論理回路65を用いて実行されることも可能である。この発明は、データの本来の本質に無関係なため、種々の形式の使用に対して効果的である。つまり、バイナリ形式で表現される全てのデータにおいて、この発明の圧縮方法は情報の損失なしに、データのサイズを小さくするのに効果的である。

次に、図2について言及すると、一般的なレベルにおいて、この明細書に開示された全ての圧縮方法は図2のフローチャートに示される。初めに、圧縮されるデータのソースが配置され、そのデータの第1ブロックがデータのソースから読み込まれる（ステップ68）。データの第1ブロックは、開示された3つの圧縮ルーチンのうちのどれか1つを用いて圧縮される（ステップ70）。圧縮されたデータは蓄積され（ステップ72）、データのブロックがまだ存在するならば（ステップ74）、それらのデータのブロックは検索され（ステップ76）、ステップ70と72を通して処理される。圧縮されるデータのソースにデータのブロックが存在しないならば、圧縮されたデータファイルが要求されたサイズ以下であるかどうかを判定される（ステップ78）。もし更にデータ圧縮が必要ならば、3つの圧縮ルーチンのどれか1つを使用して更に圧縮が可能かどうかのチェッ

クが成される（ステップ 80）。更なる圧縮が 3 つの圧縮ルーチンの 1 つを用いて可能であると判定した場合、出力ファイルをソースファイルと再定義し（ステップ 82）、再び処理ステップ 68 から 74 を通るように送られる。この再帰的な処理（繰返処理）は、出力ファイルが要求されたサイズ以下になるまで（ステップ 78）、又は、データの再帰が成果のないものと判断されるまで（ステップ 80）繰り返される。再帰を繰り返した後、3 つの圧縮ルーチンのひとつがデータファイルのサイズの更なる圧縮に対して不可能となった場合、圧縮された出力ファイルはソースファイルとして割り当てられ（ステップ 83）、このソースファイルのデータを 4 番目のデータ圧縮ルーチンが処理する（ステップ 84）。4 番目のデータ圧縮ルーチンがデータを処理した後（ステップ 84）、処理結果

のデータは要求されたサイズ以下かどうかをチェックされる（ステップ 86）。要求を満たしていれば、処理は終了する（ステップ 88）。更なるデータ圧縮が必要ならば、再びステップ 82 を通るよう送られ、ソースファイル＝出力ファイルとしてセットされる。そして、再び処理ステップ 70 から 76 を通るよう送られる。78 から 86 のステップの通り再帰される必要性があるなら、それらのステップを通るように送られる。

この発明の概略について説明してきたが、次にこの発明の詳細を説明する。開示された全てのデータ圧縮の重点と大きな特徴は、それらの方法論が動作する再帰的な本質に関係している。この方法論の再帰的な本質は、この方法が圧縮されるデータ上でいくつものパスを操作することを可能とすることである。実質上、この発明の圧縮方法論は前のパスで失われたデータを圧縮することが可能とされる。この再帰的な特徴は、冗長な情報の配置及び圧縮において、この方法論を非常に効果的なものとする。

この方法の大きな特徴は、ブロック 70 の 3 つの圧縮ルーチンとブロック 84 の圧縮ルーチンとを区別することである。ブロック 70 で設定された 3 つの圧縮ルーチンは適度にランダム化された入力データを圧縮するのに効果的である。しかしながら、ブロック 70 の方法は、各再帰によって、圧縮されたデータのランダムさをより高レベルにする傾向がある。従って、ブロック 70 の圧縮方法は、

何度かの再帰後には、ソースファイルのサイズを更に縮小することに対して非効率的になる。従って、高いエントロピーデータの圧縮に有効なブロック 84 の圧縮方法が必要となる。

ブロック 80 で成された判定は、前の再帰の成功の度合いに幾分か基づいている。例えば、最後の 5 回の再帰の履歴は、ソースファイルを通る各パスで起こったデータ圧縮度を反映して記録される。その記録に十分な進展が見られれば、ルーチン 84 を実行する必要はない。しかしながら、十分な圧縮がもはや達成されなければ、ブロック 84 の方法が実行される。

ブロック 84 の方法の重要な点の 1 つは、エントロピーが高いデータを圧縮する能力である。ブロック 84 の方法を用いてソースファイルを通るパスを作成した後、結果出力ファイルはエントロピーが高くない、という事態が発生する。このような場合には、3 つの圧縮方法 70 の 1 つを通るように戻せば、更なる圧縮結果が得られる。このような場合でなければ、ブロック 84 の方法はその出力ファイルを再帰することが可能である。

次に、図 3 A、3 B、3 C について言及する。図 2 のブロック 70 で言及されている 3 つの圧縮方法としては、ニブル圧縮 90、分配圧縮 92、直接ビット圧縮 94 がある。ニブル圧縮 90 と分配圧縮 92 は、類似した方法で作動し、両方ともコード化（エンコード）法を使用して、選んだデータ列の中で均等でないバイアスを作成する。データ列を作成後、バイアスされた列は、ニブル圧縮 90 と分配圧縮 92 の圧縮エンジンの心臓部を構成する変換／暗号化ルーチン 98 に渡される。バイアスデータを設定するために、ニブル圧縮 90 はニブルをコード化する方法 96 を用いて STRING1\$ から STRING7\$ までの 7 つのデータ列を作成する。これと対照的に、分配圧縮 92 は分配コード法 100 を用いて TOTCONT\$ と TOTMPV\$ の 2 つのバイアスされた情報の列を作成する。直接ビット圧縮 94 は、ニブル暗号化 90 又は分配圧縮 92 に共通しない方法論を使用する。

この発明の圧縮方法論は次を以下に説明する。図 2 のブロック 70 の中の 3 つの圧縮ルーチンはニブル圧縮 90、分配圧縮 92、直接ビット圧縮 94 の順番で説明される。ブロック 84 のランダム化されたデータの圧縮ルーチンを、3 つの

方法論90、92、94の各々との関連で説明する。

1. ニブル圧縮

図2、3A、4について言及すると、ニブルコード化法96は図4の表を用いることによって最も良好に説明される。この発明のニブルコード化の方法論によつ

て実行された第1のステップは、データの第1ブロックを読み込み（ステップ68）、ブロックをニブル毎に分析する。各ニブル値に従って、1つ又は、それ以上の予め定義されている制御ワード（ビット又はビット列）を、少なくともSTRING1\$、STRING2\$、STRING3\$、...、STRING7\$のうちの1つの内容に関係付ける。例えば図4について言及すれば、入力ファイルの初めのニブルが3という値を持っていれば、（すなわち、バイナリ=0011）、STRING1\$とSTRING2\$の列はそれらの列の既存の内容に関係付けられた文字0を持ち、STRING3\$は既存の内容に付加された文字11を有し、STRING4\$、STRING5\$、STRING6\$、STRING7\$は従前状態から不変である。STRING1\$からSTRING7\$の列は入力データ列から分析される各ニブルに対して作成される。より明確にするために、ニブルコード化（エンコード）法96によって達成しようとすることを図5Aと5Bを示す。

図5Aと5Bは、16ヶのニブルからなる入力列（図5Aと5Bの上部に横に沿って示されている）が図4の表中で示されているニブルコード化方法論によって処理された時のSTRING1\$からSTRING7\$の列を表している。図5Aと5Bは、64ビットと50%のバイアス（0と1が均等に分布している）を持つように設定された仮定の入力列に対して、列STRING1\$、STRING2\$、STRING3\$、STRING7\$は1と0の比率が50対50であるということを示している。しかし、列STRING4\$、STRING5\$、STRING6\$は元の入力列の50%というバイアスとは違う比率でオフセットされている。入力列の50%のバイアスが非冗長と同意ならば、入力列（図5A、5B中で示されている）は64ビット以下には縮小されない。ニブルコード化方法論はビットの数を減少させないが、非均等にバイアスされた3つの列を作成する。これらの3つの列が変換／暗号化ルーチン98に送られる時、ルーチン98はデータを圧縮することによって、3つのそれぞれの列の長さを短縮する

。ニブルコード化方法論において、バイアスされていない入力データ列からバイアスされたデータの列を作成する能力はこの発明の鍵となる特徴である。図5の”総ビット数”という欄から、ニブルコード化法96は総ビット数を減少させていないということがわかる。例えば、図5Aと5Bは、ニブル暗号96は0から15の値を持つ16のニブルをコード化するために66ビット必要としていること

を示している。2進数の形式でこれらの16のニブルの値をコード化するためには64ビット必要である。結果として、ニブルコード化法を用いれば図5Aと5Bの入力列のコード化は更に2ビット必要とする。しかし、列STRING4\$, STRING5\$, STRING6\$がバイアスされるため（同等に分配された入力データに対して）、理論上2.19、6.27、1.41ビットの節約がそれぞれに対して得られる。この表作成をサポートする計算を表1に示す。

表1				
列	$nCr = \frac{n!}{r!(n-r)!}$	nCrをコード化する為に必要なビット数	元の列の長さ (ビット)	節約できる ビット数 (ビット)
STRING4\$	8C3 = 56	5.81	8	2.19
STRING5\$	5C1 = 5	2.33	5	2.67
STRING6\$	3C1 = 3	1.59	3	1.41

入力列の長さの合計 = 64.00 ビット

出力列の長さの合計 = 66.00 ビット

節約分との差 => 2.19+2.67+1.41 = 6.27 ビット

59.73 ビット

2進数での節約 = 64-59.73 = 4.27 ビット 又は 4.27/64 = 6.67%

列STRING4\$, STRING5\$, STRING6\$は変換/暗号化方法論98を通るように送られ、ニブル圧縮90によって達成された効果的な圧縮は6.67%に近づく。勿論、

これには、観測されるオーバーヘッドレジスタや他の”資源を守る”情報は考慮されていない。しかし、データ圧縮アプリケーション内で大量のデータを処理する時は、そのオーバーヘッドは無視してよい傾向にある。

ニブルコード化法96の2つの重要な特徴はSTRING1\$からSTRING7\$の列が容易にパック又はアンパックされることと、元の入力列が7つのコード化された出力列より容易に復元（デコード）されることである。これら2つの特徴を以下に述べる。

ニブルコード化法によりコード化された列の復元

図5A、5B、6について言及すると、ニブルコード化（エンコード）法96を通して処理された入力列の復元は、次の図6の復元木（デコードツリー）に示される手順によって達せられる。どのようにして図6の復元木が使用されるかの一例を図7Aから図7Eまでに示す。図7Aにおいては、図4のニブルコード化法に4つのニブルの入力列が与えられてる。これは、図7Aに示すSTRING1\$からSTRING7\$の列を構築する結果となる。その後、図6の復元木に示された処理はSTRING1\$とSTRING2\$の列の初めの1ビットとSTRING3\$の列の初めの2ビットに適用される（図7B参照）。その結果として生じるビットの並びは、元の入力列の初めのバイトは”0001”と等しいことを示している。これは図7Aに示されるニブル1の値と一致する。これらのSTRING1\$、STRING2\$、STRING3\$の列からなる4ビットは除かれ、残った右側のビットは全て左側に適切に移され、この処理が繰り返される。図7C、7D、7Eはどのようにして元の入力列のニブル2、3、4が図6の復元木を用いて実際に復元されるかを示している。

ニブルコード化のパッケージシーケンス

7つの列をニブルコード化法96によりコード化する特有の方法により、データが適切なシーケンスにてパッケージされたかどうか容易に確認される。このシーケンスを図8に示す。

図8について言及すると、ニブルコード化法96により作成された7つの列が図8に示す方法で連結されたとすると、それらは容易に分離される。連結された

列を分離するためのアルゴリズムを図9に示す。図8、9について言及すれば、図8の連結した列を分離するためには、元の入力列におけるビット数か又はSTRING1\$の元の長さが既知の値でなければならない。これは、連結した部分を分離するために、それらの列の外に蓄積しておかなければならない情報である。図9に示すように、STRING1\$のビット数を知れば、連結した列からSTRING1\$を分離することができる。列STRING1\$の0の数がSTRING2\$の長さを生成する。列STRING2\$の長さがSTRING3\$の最初の部分の長さを生成する。列STRING1\$の1の数がSTRING4\$の長さを生成する。この処理が図9に示されるように繰り返され、図8に示す1つの列を作るために連結された元の7つの列を作成する。

列STRING4\$、STRING5\$、STRING6\$はニブルコード化法96により復元されると、これらは変換／暗号化ルーチン98に圧縮のため通される。

変換／暗号化ルーチン

図10について言及すれば、変換／暗号化ルーチン98は2段階で実行される。第1に、変換／暗号化ルーチン98に送られた列は、その列中で文字1が文字0より優位かどうかについて判定される。1が優位ならば、その列は1が優位なルーチン104に送られ、処理される。1が優位でなければ（すなわち、0が優位かまたは同等）、入力列は0が優位なルーチン106に送られる。1が優位なルーチン104と0が優位なルーチン106は類似した方法で作動し、両方とも3つの出力列を作成する。3つの列内の少なくとも1つの列は、データ列の圧縮を行うアルゴリズム108によって処理される。0が優位なルーチン106を表2を参照して説明する。

表 2				
0 が優位 (又は同等) な場合				
一度につき 2 ビット分析 される入力列	出力列	TC1\$	TC2\$	TC3\$
00		0	0	X
01		X	1	1
10		X	1	0
11		1	0	X
x = 変化無し				

入力列が、0 が優位なルーチン 106 を通って処理される時、入力列は 1 度に 2 ビットずつ分析される。分析された 2 ビットの値は判定され、表 2 に示すように、文字 0 又は文字 1 が 3 つの出力列— TC1\$、TC2\$、TC3\$ —のうちの 2 つに付加される。よって、0 が優位なルーチン 106 は入力列を処理して 3 つの出力列を作成する。

1 が優位なルーチン 104 は、表 3 に示される表に基づいて 3 つの出力列に値を割り当てることを除いては、0 が優位なルーチン 106 と類似した方法で動作する。

表 3				
1 が優位な場合				
一度につき 2 ビット分析 される入力列	出力列	TC1\$	TC2\$	TC3\$
00		1	0	X
01		X	1	1
10		X	1	0
11		0	0	X
x = 変化無し				

表 2、3 に示される、0 が優位な方法と 1 が優位な方法の特徴は、3 つの出力列から入力列を容易に再構成 (復元) できることである。表 2 を用いてコード化された情報を復元するための復元木は図 11 に示される。図 11 に従って、TC2\$ の初めのビットが 0 か 1 であるかが判別される。それが 0 ならば、入力列において元の 2 ビットが "00" か "11" であるかを判別するために TC1\$ の初めのビ

ットが判別される。TC2\$の初めのビットが1ならば、入力列において元の2ビットが"10"か"01"であるかを判別するためにTC3\$の初めのビットが判別される。TC1\$、TC2\$、TC3\$の内容を知ることによって、容易に入力データ列を再構成することができる。

表2に示される0が優位な方法のもう1つの特徴は、TC2\$の総ビット数が、入力列の全ビット数の2分の1を示すことである。従って、TC2\$のビット数を知ることにより、その2倍である入力列の長さを知る。列TC2\$における0の数は、TC1\$におけるビットの数を決定し、TC2\$における1の数はTC3\$におけるビットの数を決定する。この特徴は圧縮された情報をパッケージすることに関して、後述するように非常に貴重なものである。表2に対する復元木は図12に示され、図11の復元木と同様に作用する。表2を用いて説明された全ての利点は、表3の方法にも同等に適用される。従って、説明は繰り返さない。一例を図13Aから13Gに示す。その中では表2に示す方法論を用いて、入力列が3つの列に変換されている。図13Bから13Gは、図11の復元木を用いて、3つの列がどのようにして元の入力列に再構成されるかを示している。

図13Aから13Gについて言及すれば、入力列は表2の変換方法論を用いて処理され、TC1\$、TC2\$、TC3\$の3つの列を作成する(図13A参照)。図11の復元木はTC2\$の初めのビットに適用される。初めのビットが0なので、図11の復元木は、TC1\$の値を判別するために、TC1\$の初めのビットを見るように指示している。TC1\$とTC2\$に対する"00"という値は、"00"の値の入力列を作成する(図13B参照)。これらのTC1\$とTC2\$の2つのビットは除かれ、それらの列の残りが左に1ビットずつ移され、残りのビットがなくなるまでこの処理が繰

り返される(図13C~13G参照)。0が優位なルーチン106と1が優位なルーチン104のどちらもデータを圧縮しないことに留意することが重要である。TC1\$、TC2\$、TC3\$の長さを合わせたものは入力データの元の長さと同じに等しい。しかし、ルーチン104と106は0にバイアスされたTC1\$に対して効果的であるし又特定のビットパターンの入力列に依存するいくつかの場合において、0にバイアスされたTC3\$に対して効果的でもある。一般に、TC2\$の中のデータは均

等バイアスの傾向が強い。

図10、14について言及すれば、変換／暗号化アルゴリズムは決められた順序による0が優位なルーチンと暗号化ルーチンの実行から構成される。0が優位なルーチンについては検討済みなので、更なる説明は不要である。しかし、変換／暗号化アルゴリズム108について十分に理解される前に暗号化ルーチンを説明しなければならない。以下に暗号化アルゴリズムについて説明する。

暗号化アルゴリズム

暗号化アルゴリズム処理は表4との提携により最も深く理解される。

表4				
暗号化アルゴリズム				
一度につき 2ビット分析 される入力列	出力列	TE1\$	TE2\$	TE3\$
00		0	X	X
01		1	0	0
10		1	1	X
11		1	0	1
x = 変化無し				

表4の使用法は表2と表3が使用された方法と非常に類似している。暗号化アルゴリズムは一度に2ビットの入力列を分析することにより作動する。分析され

た2ビットの値は、1ビットが3つの出力列の1つに連結されるかどうかを判定する。また、分析された2ビットの値は、連結されるビットの値も決定する。入力列に暗号化アルゴリズムを適用した結果が出力列TE1\$、TE2\$、TE3\$の生成である。3つの出力列が既知であれば、入力列は容易に再構成される。この再構成は図15の復元木を適用する。図15の復元木の適用は以前に説明した図11、12の復元木が適用される方法と同一であり、それに応じた図15の復元木の適用の詳細な説明は不必要である。

表4の暗号化アルゴリズムの適用の一例を16Aから16Fに示す。図16では10ビットの入力列が、出力列TE1\$、TE2\$、TE3\$を作成する表4の暗号化アルゴリズムによって処理される。これらの3つの列は図15の探索木に従って処理

され、10ビットの入力列を正確に再生する（図16Bから図16F参照）。表4の暗号化アルゴリズムの重要な面は、0にバイアスされたデータを圧縮する能力である。例えば、図16Aにおいて入力列は大きく0にバイアスされる（80%の0）。これは暗号化アルゴリズムに対して理想的なバイアス配列であり、このアルゴリズムは10ビットの列を7ビットに短縮する（30%の短縮）。この例より、入力列が暗号化アルゴリズムによって処理される場合、入力列が0にバイアスされていれば、結果データの3列が圧縮されることは容易に理解される。暗号化アルゴリズム（ニブルコード化のような）と提携して作動するルーチンを0に大きくバイアスされたデータの列に適用することができれば、暗号化アルゴリズムはデータ圧縮に対して非常に効果的である。表4の暗号化アルゴリズムより、そのアルゴリズムが1にバイアスされた入力列を処理すれば、そのアルゴリズムは入力データを増大させ、データの全長を増大させてしまうことは明らかである。例えば、図16の10ビットの入力列の1の補数が暗号化アルゴリズムによって処理される場合、列TE1\$、TE2\$、TE3\$の長さの合計は15ビットとなる（150%に増大）。データの増大は圧縮ルーチンと関連する好ましくない特徴であるため、十分なソフトウェアトラップはそのような状況の発生を防ぐために設置されなければならない。そのようなトラップは図2のフローチャートのブロック80と提携して以前に説明された。

暗号化及び変換アルゴリズムについて個別に説明したので、この発明の再帰的な本質の一面を示す、暗号化方法論と変換方法論とを結合した複数のアプリケーションによるいくつかの方法を以下に示す。任意のデータセットに繰り返し適用される暗号化／変換アルゴリズム108の能力は、この発明の圧縮方法論が力強いデータ圧縮”エンジン”を形成することを可能にする。任意のデータセットを再帰する変換／暗号化アルゴリズム間の相互作用は次に示す例によって最も容易に理解される。

変換／暗号化アルゴリズム使用時に用いられる再帰的方法論

図3A、5A、5B、10、17A、17Bについて言及すると、ニブルコード化法96は、STRING1\$、STRING2\$、...、STRING7\$を作成する仮定の入力デー

タに適用される。この仮定の場合において、STRING4\$が大きく0にバイアスされているため、その列が変換／暗号化ルーチン98によって処理される理想的な候補を形成することを想定してみよう。この仮定の列は選ばれ、変換／暗号化ルーチン98を通して、0が優位なルーチン106を通される。図17A、17Bは仮定の入力列STRING4\$における変換アルゴリズムの動作を表現している（以下、変換は表2の0が優位なルーチンと同義語である）。図17の右側の欄から明らかにされるように、仮定の入力列は64ビットで0に78%強バイアスされている。入力列上で作動する0が優位なルーチンは、3つの出力列TC1\$、TC2\$、TC3\$を作成する。以前に説明したとおり、0が優位なルーチンは圧縮アルゴリズムではないので、データの全ビット数を減少させない。しかし、そのルーチンは少なくとも1つは0にバイアスされた列を作成するよう動作する。例においては、0にバイアスされた列はTC1\$であり、95%のバイアスを帯びている。図18について言及すれば、仮定の入力列STRING4\$を含む64ビットを処理し、3つの出力列TC1\$、TC2\$、TC3\$を作成する。

図19、20について言及すれば、TC1\$はいくつかの出力列の中で最も高く0

にバイアスされているので、変換／暗号化アルゴリズム108を通るように処理されるのに3つの列の中で最適なものとなる。アルゴリズム108を通るTC1\$の処理は、図19に示され又図20に解析的に示されている。アルゴリズム108を通して列を処理するために使用される1つのその方法は、初めに入力列を表4の暗号アルゴリズムを通して処理し、データ圧縮の成果があるかどうかの判定をする。例えば、暗号化アルゴリズム108を通るTC1\$の初めのパスの間で、3つの列TE111\$（長さ10ビット）、TE211\$（長さ1ビット）、TE311\$（長さ1ビット）が作成される。これはTC1\$の全長を8ビットだけ減少させる。長さの減少が起こらなければ、又はデータの増大が起こったら、データ列TC1\$を暗号化（コード化）することは有益ではなく、表2の変換アルゴリズムを適用する。1つの可能な変換／暗号化アルゴリズム108はまず初めに暗号化アルゴリズムを用いて入力列を処理し、データ圧縮の成果があるかないかを判定する。圧縮の成果がなければ、列は変換される（0が優位な方法論を使用して）。この特別な方法を”

処理及び判定”方法として呼ぶこととする。”処理及び判定”方法を出力列に適用し続けることにより、各出力列TE111\$、TE211\$、TE311\$を入力列にして、暗号化アルゴリズムを通して処理する。図19に開示されるように、TE111\$の暗号化はデータを4ビット縮小させる。列TE211\$に暗号図を適用すると、暗号化ルーチンは単一ビット列上で動作することがわかり、この”レッグ”の更なる処理を控える。いくつかの例において、暗号化アルゴリズムの動作は列の増大を引き起こしている。これらの場合は暗号化（コード化）せず、そのレッグの更なる処理を控える。TE311\$は1ビットなので、それに対する更なる処理は控える。

再び、”処理及び判定”方法をTE111\$の暗号化により作成された3つの列に適用すると、TE121\$の暗号化はデータを2ビット縮小させる。この特定の暗号化ルーチンのアプリケーションにおいて、列TE121\$はビット数が奇数なので、1ビット（図19参照）の残余ビット（残り）がある。これは、奇数ビット数の列が暗号化ルーチンか又は変換ルーチンで処理される場合、常に起こる。”処理及び判定”方法の再帰的な列TC1\$への適用において説明した工程を図20に示す。再帰的な”処理及び判定”方法の列TC1\$への適用における最終結果はTC1\$を8ビッ

トの長さに縮小、又は60%の縮小である。

図18、21について言及すると、TC1\$の圧縮時に用いられた方法と同じものをTC2\$に適用する。特に、まず暗号化アルゴリズムは圧縮成果があるかないかを判定するため適用される。TC2\$の例では、32ビットの列TC2\$への暗号化ルーチンの適用は、圧縮の成果がない（暗号化ルーチンで作成された3つの列の全長さは入力列の長さと同じ）。従って、TC2\$を暗号化せず、変換アルゴリズムを適用する。前述のように、変換アルゴリズムは圧縮アルゴリズムではなく、出力列の少なくとも1つが0のバイアスを帯びるようにするツールとして役立つ。列TC2\$の変換は3つの出力列TE112\$、TE212\$、TE312\$を作成する。データ圧縮が可能かどうかを判別する”処理及び判定”方法を使用して、順番にこれら3つの列は判定される。初めに、列TE112\$は、連結した長さがTE112\$の長さと同一な3つの列に暗号化される。この点では、2つの”変換”が並んで実行される（元の列STRING4\$の変換と列TC2\$の変換）。結果としての3つの列は圧縮の成果があるかど

うかを判定するために暗号化される。”処理及び判定”方法の手順に従って、初めに列TE122\$の暗号化を試行するが、これは1ビットの増加を生じさせる。列TE122\$の追跡は成果がないと想定し、変換アルゴリズムをその列に適用すると、4ビットの列を更に圧縮することは不可能なことに気付く。よって、列TE122\$を更に圧縮することは不可能である。

暗号化された列TE122\$が暗号化されると(図21参照)、1ビット増加することに気付くことが重要である。盲目的に暗号化アルゴリズムを適用することは、データの増加を引き起こす可能性がある。列TE212\$の分析と列TE312\$の分析は図22、23に各々示される。全ての縮小が合計されると、”処理及び判定”方法は列TC2\$長さの7ビット又は約21%の縮小という結果となる。図24に示すように、”処理及び判定”解析を列TC3\$に適用すると、列TC3\$の長さにおいて3ビットの縮小が得られる(又は25%)。以上のことより、変換/暗号化アルゴリズム108が”処理及び判定”方法の形式を取った時、初期の列STRING4\$の有力で総合的な圧縮が得られる。この例において、列STRING4\$を22ビット又は34

%近くだけ縮小させることができる。

変換/暗号化アルゴリズム108の実行時に”処理及び判定”方法を用いることは十分なデータ圧縮を作り出すけれども、その方法の実行は、単純ですっきりしたプログラミング技術には成り得ない。これは非常に本質的なことである。なぜなら、”処理及び判定”方法は、各方式の処理での圧縮結果を最適化するように設計されているからである。その方法はデータを高速に圧縮するけれども、暗号化又は変換されたデータを復元する時、全ての分岐において成された判定(すなわち、変換又はコード化するために)は再呼び出しされる。その情報を記録する技術は容易であるが、それはソフトウェアに望ましくない複雑さを与える。加えて、各ノードでのその最適化は不必要であると考えられている。なぜなら、圧縮可能なデータが、変換/暗号化ルーチン98の初めの実行によって圧縮されないならば、そのデータは次のパスで圧縮ルーチン70(図2参照)によって繰り返して処理される時、データの次に続くパスの中の1つで処理される。よって、再帰はこの発明において2つのレベルで発生する。第1のレベルでは、再帰は変

換／暗号化アルゴリズムによって実行される複数の変換／暗号化アプリケーションにおいて発生する。第2のレベルでは、再帰はデータを必要なサイズ以下に縮小する圧縮ルーチン78の複数のアプリケーションによって、発生する。この見解によれば、変換／暗号化アルゴリズム108を用いた”処理及び判定”方法は非最適化方式より優れた圧縮結果を達成するとは考えられていない(”処理及び判定”方法は高速に結果を出すけれども)。より時間をかければ、簡易化された変換／暗号化アルゴリズムが使用される。簡易なアルゴリズムを以下に開示する。

この簡易化した技術を説明することにおいて、図19から図24に図示された例の説明を再検討することは有益である。これらの図より、最も大きい圧縮成果が最左方の分岐から集められているということが一般的に観察される。この傾向はほとんど全てのレベルで観察される。例えば、列TC1\$、TC2\$、TC3\$の中で最も大きい総合的な圧縮は列TC1\$による、60%の縮小だった。TC1\$は列STRING4\$から作成された最も左方の列である。次に、一般的に圧縮／暗号化アルゴリズムに

よる圧縮時において、最も右側の列が中心の列よりも成果があることが観察される。これらの傾向は驚くべきものではない。前述のように、変換ルーチンは可能な限り0を多く含むTC1\$（最も左方の分岐）を”ロード”するように設計されている。また、TC2\$は50%のバイアス度に近づく傾向にある。最も左方の分岐は、より多くの0を含む傾向にあるため、暗号化アルゴリズムはこれらの列に対して最も効果的となる。

この考え方から、1つのより簡易な変換／暗号化アルゴリズムは分析実行時、最も左方の枝を見る。例えば、変換してコード化する判定を常に実行できるが、変換後、最も左方の枝だけコード化することもできる。次に、第1の変換をして、第1の変換の最も左方の枝について第2の変換をした後、暗号化（コード化）をする。しかし、第2の変換の最も左方の枝だけを暗号化することもできる。より洗練された方法は、ブロック毎のデータの分析と、その分析に基づいて最適化された変換／暗号アルゴリズムを判定しデータ分配パターンを選ぶことである。各最適化されたアルゴリズムはあるワード値を与えられ、そのワード値は入って

くるデータの最初のワードとして蓄積される。この目的の為に2ビットのワードを作成することにより、4つの最適化されたアルゴリズムが有益である。3ビット用いることにより、8つの最適化されたアルゴリズムがコード化される。

列STRING5\$、STRING6\$がSTRING4\$に対して示した方法と同様にコード化／変換アルゴリズムを通ることは明白である。列STRING5\$、STRING6\$は1より0を多く含んでいるので、圧縮／暗号化アルゴリズム108が、これらの列の長さの圧縮に対して成果があることが期待される。ニブル圧縮ルーチン90の1つのパスの完了により、4つの圧縮されない列（STRING1\$、STRING2\$、STRING3\$、STRING7\$）と3つの圧縮される列（STRING4\$、STRING5\$、STRING6\$）が作成される。圧縮された列は1連の可変長ビット列の数により表され、その各列は1つ又はそれ以上の暗号化アルゴリズム及び変換アルゴリズムの適用することによって作成される。ニブル圧縮ルーチン90の1つのパスで作成される種々の列と列の部分が、どのようにして後の元の入力列の再構成に使用されるようにパックされるかの一例を次に示す。

図3A、8、9、25について言及すれば、この例の目的に対して、ニブルコード化法96がSTRING4\$を（この列のみ）圧縮のために変換／暗号化ルーチン98に通したと想定しよう。そして、変換／暗号化ルーチンに通される1つの暗号化データを単純に処理する非常に簡易な暗号化／変換アルゴリズムを想定しよう。図25において、暗号化アルゴリズムが32ビットの入力列を処理し、TE1\$、TE2\$、TE3\$から成る33ビットの出力列を作成している。傍注として、入力列において十分な0のバイアスがなければ、暗号化ルーチンがデータを1ビットだけ増大させる。マイナスの結果が発生するが、この出力列の圧縮の例は不変であり、弱まりもしない。

図25において、出力列TE1\$の総ビット数（16ビット）が入力列の総ビット数の1/2となっている。これは入力列が偶数ビット数である限り常に起こる現象である。入力列が奇数ビット数の場合は残余が生じる。この場合を図26の例に示す。

TE1\$における1の数が列TE2\$の総ビット数を決定する。更にTE2\$の0の総数が

TE3\$の総ビット数を決定する。3つの列をTE1\$/TE2\$/TE3\$の順序で連結すると、元の入力列の長さを知ることにより、それらは容易に互いから分離できる。例えば、元の入力列が32ビット長であることが既知の場合において、まず初めの32ビットを、連結された列から取り出す。これがTE1\$の列になる。列TE1\$の1の数を計数(カウント)する。その数から、変数TE2\$を形成するために、連結された列から次に取り出すべきビットの数が決定される。次に、列TE2\$における0の数が計数され、それによりTE3\$に割り当てられるビットの数が決定する。この例により、暗号化ルーチンによって作成された3つの出力列は他の列との関連で特有に定義されており、それらの列は所定の順番で連結されていて、その結果、元の入力列の長さを知ることにより、個々に識別され、連結から分離される。従って、この発明の暗号化ルーチンは0のバイアスデータの圧縮のための力強いツールだけでなく、連結が容易に解除される出力列を作成する。

暗号化ルーチンが図25の連結された出力列を作成すると、この列を図8の列のパッケージシーケンスに示されるSTRING4\$の代わりとする。従って、図9の復元木の実行時、STRING4\$の0又は1の数が必要になった適当な時点で、コード化された列STRING4\$はその構成要素の列(TE1\$、TE2\$、TE3\$)に分離され、図15の復元木に従って復元される。これは元の入力列の再構成であり、図9の復元木はニブルコード化法で用いられた列の連続鎖の復元を実行できる。

図26について言及すれば、第2の例では、第1の例で用いられたものより複雑なコード化/変換アルゴリズムが用いられる。第2の例では変換ステップを用い、そして暗号化ステップを用いる。この第2の例を図26を参照して説明する。

まず初めに、32ビットの入力列を変換ルーチンを使用して処理し、それによりTC1\$、TC2\$、TC3\$を作成する。TC2\$の総ビット数は入力列の総ビット数の1/2である。これは入力列が奇数のビット数でなければ、常に起こる状況で、入力列が奇数のビット数の場合、残余が生じる。列TC2\$での1の数を計数することにより、TC3\$の総ビット数を得る。更に、列TC2\$での0ビットの数はTC1\$の総ビット数を直接表す。この例において、TC1\$を取得し、暗号化の入力列として使用する。

る。TC1\$は奇数のビット数なので、値が0の残余ビットが生じる。この残余ビットは記録され、暗号化アルゴリズムは、通常通りTC1\$を暗号化（コード化）し、残余ビットが存在しないものとして扱うよう実行される。最終的な出力列を図26に示す順序で作成すれば、元の32ビットの入力列は図25に示されるものと類似した技術を用いて容易に再構成される。

定義によって、元の入力列（32ビット）の長さは既知のもので、そのビット数の $1/2$ を出力列から取り出し、TC2\$を生成する。TC2\$の1の数は、出力列から取り出すTC3\$の長さを決定する。TC2\$の0の数を計数し、その数を半分に割ることによって（小数部は除く）、TE1\$の総ビット数が決定される。TC1\$は奇数のビット数なので、TE1\$に割り当てられた4ビットの後に残余ビットが存在する。

その残余ビットを除去し、列TE1\$の最後に連結する。TE1\$の1の総数によりTE2\$の総ビット数が決定され、TE1\$の0の総数によりTE3\$の総ビット数が決定される。TE1\$、TE2\$、TE3\$はTC1\$を再構成するために図15の復元木を用いて結合され、その後TC2\$、TC3\$、TC1\$は元の入力列を再構成するために、図7の復元木に従って結合される。

図27A、27B、27C、28について言及すれば、変換／コード化アルゴリズムの第3の例を説明する。この例では、変換／変換／コード化／コード化技術が用いられる。種々の変換／コード化ステップが図27Aに示される”ツリー”ダイアグラムに従って元の入力列に適用される。第1と第2の例を用いて説明された解析は、この第3の例に適用される解析と同一のものである。変換／変換／コード化／コード化アルゴリズムのアプリケーションにおいて作成される全ての列及び列の構成要素は、1つの出力列を形成する図28の方法において連結される。元の46ビットの入力列を再構成する時、出力列は以下に示す方法を用いてその構成要素列に分離される。まず初めに、定義により入力列の長さは既知なので、その入力列の長さの $(1/2) \times (1/2)$ （すなわち $1/4$ ）と等しい列G5の長さも既知である。入力列は46ビットなので、列G5は11.5ビットである。少数部分を除去することにより、GS5の長さは11ビットとなる。出力列から初めの11ビットを取り出し、その中の0を計数することにより、列GGS

4の長さを決定する。処理は以下のように進む。

- ・ 列GGS4の1を計数することにより列GGS5の長さを決定する。
- ・ 列GGS5の0の数を計数することにより列GGS6の長さを決定する。
- ・ 図15の復元木に列GGS4、GGS5、GGS6を適用することにより、列GGS4を再構成する。
- ・ 図11の復元木に列GGS5、GGS6、GGS4を適用することにより、

列SON2を再構成する。

- ・ 列SON2はビット数が奇数なので、終端の残余ビットを取り出す。
- ・ SON2の0の数を計数することにより列SON1の長さを決定する。
- ・ SON1の長さを $1/2$ に分割することにより列GS2の長さを決定する。
- ・ GS2の1の数を計数することにより列GS3の長さを決定する。
- ・ GS2の0の数を計数し、その数を $1/2$ に分割することにより列GGS1の長さを決定する。
- ・ 図15の復元木に従って列GGS1、GGS2、GGS3を結合することにより、列GS1を再構成する。
- ・ 図11の復元木に従って列GS1、GS2、GS3を結合することにより、列SON1を再構成する。
- ・ 図11の復元木に従って列SON1、SON2、SON3を結合することにより、元の入力列を再構成する。

次に図3Aについて言及すると、この発明のニブル圧縮方法論90を要約すれば、ニブル圧縮方法論90はニブルコード化法96と変換／暗号化ルーチン98から構成されるという説明がされている。ニブルコード化法は7列のデータを作成するのに効果的である。その列のいくつかは0にバイアスされている。変換／暗号化ルーチン98はニブルコード化法96によって作成され、0にバイアスされた列の圧縮に効果的である。種々のコード化技術がニブルコード化法96と変換／暗号化ルーチン98の両方において使用され、それらによって作成された列

は、隣接する列の長さの情報を持つ。このコード化技術は元の入力データの容易な記憶、検索、再構成を考慮している。

図2、3Aについて言及すれば、ソースファイルにおける全ての入力データがニブル圧縮90によって処理された後、出力ファイルは要求されたサイズ以下かどうかを判定するためにチェックが行われる（ステップ78）。ソースファイルを通る1つのパスにおいて十分圧縮が実行されていれば、プログラムは終了する（ステップ88）。しかし、多くのアプリケーションにおいて、データを繰り返して処理した場合にのみ（ステップ80、82）、十分な圧縮が達せられるという場合がある。しかし、複数の再帰を圧縮対象データに適用する場合において、3つの圧縮ルーチンの内の1つがデータのサイズの更なる縮小においてもはや効果的でないという点に達する。この時点では、更なる再帰は成果がなく、ある場合においては、データを増大させてしまうかもしれない（暗号化アルゴリズムが十分に0で重み付けされていないデータに適用された場合について、以前に説明したように）。圧縮ルーチン70を更に反復しても成果がない場合、ランダム化されたデータの圧縮に対して非常に効果のあるルーチン84を用いる。このランダム化されたデータの圧縮ルーチン84を次に説明する。

図2、3A、3B、3C、29について言及すると、3つの圧縮ルーチンのどれか1つを用いて複数回反復して処理を行うと、出力ファイル中の1と0の分配が、高い率でランダム化し始める（すなわち、0と1の分配が、出力ファイルから取り出されるどんなサイズのサンプルに対しても50%に近づく）。この時点において、圧縮ルーチン70はまだ圧縮能力はあるが、効率という点で不利である。

図29の圧縮ルーチンは、高率でランダム化されたデータの圧縮に関連した特有の問題を処理するのに適している。図29について言及すれば、この特有の方法論の圧縮ルーチンは、高い率でランダム化されたデータを含むソースファイルからデータの初めのブロックの読み込みを行う（ステップ112）。この例のため、データの1ブロックサイズが100ビットであると仮定しよう。次に、そのブロックの所定の部分における0の数が計数される（ステップ114）。例えば

、100ビットの内の初めの80ビットにおいて発生する0の数を計数する。計数された0の数と、完全なランダム化（すなわち、50%の0と50%の1）が成されているという仮定の基に、どの程度の0がブロックの残りの部分に残るかわかる。これを知ることにより、ルックアップテーブルはコード化され、その中では存在する可能性がある全てのビットパターンが各々のアドレスに連結されている。ブロックの残りの部分は除去され、ルックアップテーブルのアドレスが代わりに配置される（ステップ116）。データの初めの部分とコード化された部分は蓄積され（ステップ118）、次のブロックがソースファイルから読み込まれ（ステップ122）、118を通るように処理される（ステップ114）。

ランダム化されたデータの列の圧縮は不可能であることは広く明言されている。この誤信は、多数のランダム化されたデータセットに対して、可能なビットパターンの組み合わせを計算することは困難である事実を根拠にした部分にある。例えば、100ヶの1と100ヶの0が200ビット長の1ブロック内で分配されれば、

$${}_{200}C_{100} = 9.055 \times 10^8$$

の組み合わせが可能である。その数が取り扱いにくいことは明確であり、これがランダム化されたデータ列は圧縮不可能であると言われる始まりである。この説明における、この発明の圧縮法の重要な点は、そのような大量のデータを扱わず、より小さな部分を処理するところにある。ランダム化されたデータの圧縮ルーチン84の一例を図30A、30Bを用いて説明する。

この例において、ブロックサイズは200ビット長とし、ブロック毎にデータの最後の5%（10ビット）だけ圧縮する。第1段階は初めの190ビットにおいて発生する0の数を計数することである。これを知ることにより、0と1が完全に均等に分配されていると仮定した場合に、いくつの0が最後の10ビットに

発生するはずであるかを知る。最後の10ビットの内0が9ヶなら、ルックアップテーブル1を参照する（30A参照）。ルックアップテーブル1は一意的ルックアップテーブルアドレスを、可能な各ビットパターンに関連付ける。適切なルックアップテーブルアドレスが選択され、初めの190ビットに連結される。次

の列全体が蓄積され、次の200ビットの列が読み込まれ、同様に処理される。初めの190ビットの除去後、残りの10ビットが0が8ヶのビットパターンであれば、ルックアップテーブル2（30B参照）が、前の190ビットに連結するアドレスの判定に用いられる。

種々のビットパターンに対するアドレスをコード化することによって、10.0ビットから2.0ビットの範囲で変動するビットセービングが可能である（表5参照）。

表5			
最後の10ビットにおける0の残数	ncr	アドレスをコード化 する為に必要な ビット数	節約できるビット数
0	0	0	10-0 = 10
1	10	3.34	10-3.34 = 6.66
2	45	5.5	10-5.5 = 4.5
3	120	7.0	10-7.0 = 3.0
4	210	7.8	10-7.8 = 2.2
5	252	8.0	10-8.0 = 2.0
6	210	7.8	10-7.8 = 2.2
7	120	7.0	10-7.0 = 3.0
8	45	5.5	10-5.5 = 4.5
9	10	3.34	10-3.34 = 6.66
10	0	0	10-0 = 10

セービング（節約量）を増加させるには、非圧縮のブロックの部分114を減少させ、ステップ116で処理される残りの部分をより大きくする。例えば、1ブロック200ビットのうちの最後の20ビットを用いれば（最後の10ビットの代わりに）、最悪の場合において必要とされる最大ビット数は

$$20C_{10} = 184,755 \Rightarrow 17.6 \text{ (コード化するために必要なビット)}$$

であり、最悪な場合のセービングは

$$20 - 17.6 = 2.5 \text{ ビット}$$

である。

図2について言及すれば、ランダム化されたデータ圧縮ルーチン84が全ソースファイルを1回通った後、圧縮されたデータが要求されたサイズか又はそれ以

下かどうかを判別するためのチェック 86 が再び成される。条件が満たされていれば、プログラムは終了する（ステップ 88）。条件が満たされていなければ、データは 3 つの圧縮ルーチンの 1 つ 70 を通るように送り戻されるか、又は圧縮ルーチン 84 を通るように送り戻される。データをルーチン 70 とルーチン 84 のどちらに送り返すべきかを判定するために用いることが可能な 1 つの方法は、単純にファイルのバイアスを調べることである。ファイルのバイアス率が予め設定していたレベル以下だった場合、ルーチン 84 を用いると圧縮効果はより大きくなりやすい。バイアス率が予め設定していたレベル以上だった場合、圧縮ルーチン 70 がルーチン 84 より優れた結果を出しやすい。

ランダム化されたデータ圧縮ルーチン 84 を完全にランダム化されたファイルを用いた場合でだけ説明したが、このルーチンは、各サンプルされたブロックに 1 ビットを単純に加えることによって、完全にランダム化されていないファイルにも適用できる。例えば、このビットに 1 を設定するならば、このブロックは完全にランダム化されておらず、それゆえルーチン 84 による処理は不可能であるということの意味する。このビットに 0 を設定するならば、そのブロックに含まれるデータをルーチン 84 により処理することは可能であるということの意味する。

図 31 について言及すれば、ランダム化されたデータ圧縮ルーチン 84 により圧縮されたデータの伸張（圧縮解凍）において、データの初めのセグメントが読み出される。次に、非圧縮部分の 0 の数が計数される（ステップ 124、126）。この数を知ることににより、圧縮された部分を伸張すれば、どれだけの 0 が表れる

かを計算できる。どれだけの 0 が伸張される部分の中に存在するかを知ることにより、アドレスはどれくらいの長さか、そして圧縮された部分はどれくらいの長さかをルックアップテーブル技術を用いて知る（ステップ 128）。次に適当なルックアップテーブルを参照し（ステップ 130）、コード化されたデータ部分を適当なビットパターンに置換する（ステップ 132）。このシーケンスはソースファイルの全内容が処理されるまで（ステップ 134）、繰り返される（ステ

ップ138)。

図2について言及すれば、ランダム化されたデータ圧縮ルーチン84が高率で又は完全にランダム化されたデータを圧縮するためにどのように動作するかを説明した。ファイル及びデータ圧縮を発生させるパス上を通る複数のパスを作成するために、どのようにルーチン84を用いるかについても説明した。加えて、ランダム化されたデータ圧縮ルーチン84と圧縮ルーチン70間の相互作用が示されていて、それはデータを特徴付ける0のバイアスの度合いにかかわらず、データ圧縮に効果的である。

図2、3A、3Bについて言及すれば、すでに説明したとおり、ニブル圧縮90は3つの圧縮ルーチンの内の1つでブロック70にて用いられる。次に、データ圧縮に使用するためにブロック70で用いられる第2の圧縮ルーチンについて示す。この第2のルーチンは分配圧縮ルーチンであり、それは図3Bに示される。

2. 分配圧縮

図3Bについて言及すれば、分配圧縮92はニブル圧縮90(図3A参照)を用いて示したのと同様な一般的な形式をとる。特に、ニブル圧縮90と同様に、分配圧縮は2工程の方法を用いる。すなわち、初めの未加工のデータはデータの複数の列作成のため分配コード化法により処理される。これらの列のいくつかは圧縮データ作成のため変換/暗号化ルーチン98によって処理される。変換/暗号化ルーチン98は図3Aを用いて説明した変換/暗号化ルーチン98と同一な

ので、これ以上の説明は、以前した説明の繰り返しになる。ルーチン98の説明を除いた分配コード化法100の詳細な説明を以下に示す。

図32について言及すれば、分配コード化法140は、初めに、ニブル毎に、入力データを判別することにより、最も頻繁に発生するニブル値を決定する(ステップ142)。次に、データは予め決められたニブルのセグメントに分離される(ステップ144)。セグメントの長さはある程度任意であるが、10ニブルから数百ニブルの長さで配置されたセグメントは最も実行可能な傾向にある。初めのセグメントは検索され、その内容がニブル毎に、そのセグメントのいくつか

のニブルが最も頻繁に発生するニブルの値と合致するかどうかについて判別される（ステップ146）。

図32、33Aについて言及すれば、データの流は各々が16ニブルを含む100セグメントから構成されると仮定しよう。また、最も頻繁に発生するニブル値が14（すなわち、2進数1110）であることを仮定しよう。各セグメントは最も頻繁に発生するニブルの発生に関して判別され、最も頻繁に発生するニブル値を含むセグメントに対して制御ビットCONT\$を1に設定する（図33Aのセグメント2、3、100参照）。セグメントが最も頻繁に発生するニブル値を含まなければ、CONT\$値に0を設定する（図33Aのセグメント1、99参照）。次に、最も頻繁に発生するニブル値を含む各セグメントに対して、その列の中のニブル値の配置を変数POS\$に記録し、最も頻繁に発生するニブル値は各セグメントから取り除かれる。残った”穴”は、各セグメント内において右側のほとんどのニブルを”左方に移す”ことによって埋められる（ステップ160）。表6から、POS\$値はアドレスされ、対応するMPV\$値は記録される（図33B）。次に、列は全MPV\$の列のTOTMPV\$を形成し、全CONT\$の列TOTCONT\$を形成する。最も頻繁に発生するニブル値が発生しないセグメントにおいて、全16ニブル値は0から14までの値の範囲でマップされる（ステップ152）。このマッピングは15の基本値を用いる従来のパックルーチンの適用を可能とする（パックルーチンの例に関しては定義部を参照）。

表6			
位置 (POS\$)	マップされた位置値 (MPV\$)	位置 (POS\$)	マップされた位置値 (MPV\$)
1	0000	9	0100
2	0010	10	0110
3	0001	11	0101
4	0011	12	0111
5	1000	13	1100
6	1010	14	1110
7	1001	15	1101
8	1011	16	1111

分配コード化法により実行された節約を以下に示す。

A. 最も頻繁に発生するニブル値が発生しない列においては、そのセグメントに含まれる各16ニブルに対して約1.488ビット節約できる。例えば、32ニブル長のセグメントについて処理すれば、2.976ビット節約できる。

B. 最も頻繁に発生するニブル値が発生するセグメントにおいては、2つの違う節約に役立つ。第1には、最も頻繁に発生するニブルの位置より先に発生する全てのニブルに対して、これらを先に述べた方法でパックすることができる。この方法でパックすることにより、最も頻繁に発生するニブルより前に発生する全てのニブルに対して0.093ビット節約される。第2の節約の原因は、表6にマップされた位置値に示した方法にある。最も頻繁に発生するニブル値は1回以上発生するので、それらが発生する時、そのセグメントの上位半分の位置で発生するよりも下位半分の位置で発生する傾向がある。この平均を取ると、0にバイアスされたマップされた位置値 (MPVS) を得られ、位置9から16で見つけられる0よりも多くの0を位置1から8で得られる。実際この0のバイアスが発生すれば、それは、圧縮のための変換/暗号化ルーチン98を通るように送られる

完全な候補となる列TOTMPVSに反映される。

C. を列TOTCONT\$が変換/暗号化ルーチン98を通るように送ることにより、第3の節約の原因が示される。任意の均等に分配されたデータにおいて、最も頻繁に発生するニブル値は列の約64.2%において発生し、列の約35.8%において発生しない。このことは、変換/暗号化ルーチン98を通る候補となるTOTCONT\$のバイアスに反映される。

図34について言及すれば、分配圧縮92を用いてセグメント1から100を処理した後、結果としての列は図34に示される順序に基づいて連結される。出力列を元のファイルを再構成するために復元する時、復元処理は容易である。まず、列MOD(TOTCONT\$)及び列MOD(TOTMPVS)は、変換/暗号化ルーチン98により処理された復元情報を用いてすでに説明された方法で復元される。次に、TOTCONT\$にアクセスすることにより、各パックされたブロックの状態を知る。ブロックが最も頻繁に発生するニブル値を含んでいなかったら、単純にその内容をアンパ

ックする（アンパック操作の例は定義部のパックルーチン参照）。セグメントが最も頻繁に発生するニブル値を含んでいたら、最も頻繁に発生するニブル値より先に発生するバイトをアンパックする。そこで、アンパックされた部分がブロックの残りの部分と連結される。分配圧縮92はニブル圧縮が適用されたのと同じな方法でデータのブロックに連続的に用いられる。このように、ニブル圧縮90及び分配圧縮92は再帰的であり、それらが適用される時はいつでも、データ圧縮に対して効果的である。従って、少しだけの縮小が可能な時でさえ、その少しだけの縮小は再帰により拡大される。そこでは、“単一のパス”の圧縮技術を使用した時に可能な分よりも更に大きい縮小が成される。例えば、0.093ビットの縮小でさえも100万回の再帰を実行すれば、93.0キロビットの圧縮という結果になる。

図2、3Bについて言及すれば、この部分では、分配圧縮92の処理とそれがどのようにしてデータを圧縮するためにパックルーチン及び変換／暗号化ルーチン

ン98を用いて動作するかを説明した。3つの圧縮ルーチンの3番目を次に説明する。

3. 直接ビット圧縮

図2、3A、3B、3C、35について言及すれば、直接ビット圧縮94はブロック70においてニブル圧縮90及び分配圧縮92のように動作する。従って、ソースファイル上で再帰的（反復的に）に使用できる。ニブル圧縮90及び分配圧縮92との相違点は、変換／暗号化ルーチン98を使用しないことである。

直接ビット圧縮は圧縮するために単一のアルゴリズムを用いる。このアルゴリズムは図35を用いて最も容易に説明される、直接ビットコード化アルゴリズム95である。ソースファイルは読み込まれ、そのファイルから8ビットのワードが分析される。各特有のワードの値によって、予め決められた値がワード値から引かれ、バランス値を生成する。例えば、ケース1の範囲にあるワード値は0の制御ワードが割り当てられる。ケース2の範囲にあるワード値は1の制御ワードが割り当てられる。0のワード値（ケース9）は、10000001の制御ワードが割り当てられる。図35に示すように、ケース2から8は、コード化ビット

数と制御ワードのビット数を加えると7になり、ケース1及び9は8となる。ケース2から9は分析ビットが割り当てられる。ケース1は分析ビットは必要としない。図35の右側の欄は、各ケースについて、コード化ビットの数と制御ワードのビットの数と分析ビットの数とを各ケースに対して加えるとケース9を除いては8になり、ケース9では合計が9になる。

図35、36について言及すると、直接ビットコード化法95を適用する一例において、入力列がBYTE1からBYTE5の5バイトから構成されると仮定しよう。BYTE1からBYTE5は、図36に示される各値と同等の2進数である。32のワード値を持つBYTE1はケース3の状態として配列され、そこでは、その32という値から32という値が引かれ、0のバランス値と、“11”の制御値と、1の分析ビット

が生じる。同様に、BYTE2、BYTE3、BYTE4に対するバランス値、制御値、分析ビットが図35より得られる。次に、バランス出力列 (BOUTS\$) は、バランス値をEB1からEB5の順番に連結することにより、作成される。次に、出力列CW/ROUTS\$は、EB1からEB5をその順番で連結し、その後にEB1からEB5の分析ビットを逆の順番で連結することにより、作成される。

元の列STRING\$を再構成するための (BOUTS\$) と (CW/ROUTS\$) のデコードは、図37の復元木に従って行われる。CW/ROUTS\$とBOUTS\$の最後に残った列に対しては、デコードのためには分析ビットは不要である。従って、最後の分析ビット (図36の例ではEB4\$に示される) はCW/ROUTS\$において元のワード値の再構成の可能性を失うことなく、除去可能である。よって、ファイルにおけるデータのエントロピーにかかわらず、直接ビットコード化法95を用いてファイルを反復処理する時毎に、1ビットのセービングが保証されている。

直接ビットコード化に対する1つの方法が図38Aから38Cに示される。初めに、入力列 (STRING\$) は、2つの出力列BOUTS\$及びCW/ROUTS\$を作成するために直接ビットコード化法により処理される (これは図36に示す例の結果であり、図38Aに示される再帰レベル“0”の出力を形成する)。連続的な繰り返し処理において、列BOUTS\$は、直接ビットコード化法により処理され、それにより

EB\$(CW)とEB\$(R)による列が連続的に作成される。この処理は残っている列BOUTS\$が予め決められたサイズ(32ビットのような)に縮小されるまで、続けられる。次に、図38Bの4つの列が図のように構成され、図38Cに示すように相互に連結される。更なる再帰が必要ならば、2番目の再帰カウンタは回数のトラックを保持し、38Cでの列は38Aの方法論に戻される。従って、技術に熟練した者は、適当なカウンタを保持することにより、この発明の直接ビットコード化法は、入力列のビットパターンに関係なく、入力列を1ビット縮小する効果があることがわかる。種々のカウンタ及びレジスタを保有していくことは特定の“損失”が必要だけれども、十分な大きさのファイルに対して、このオーバーヘッドは直接ビットコード化法により得られる節約と比較すると無意味である。

直接ビットコード化法において常に入力列から1ビット削り落とすという規則の例外は、入力バイトが0のワード値を持つ場合だけである。この例においては、ケース9の状態があり、そこでは、コード化ビットと制御ワードと分析ビットとが加えられ、9ビット長の列となる。この場合において、結合されたCW/ROUTS\$及びBOTSS\$の長さが事実1ビット増加する。この望ましくない結果を取り除くために、均等に分配された1024バイトのデータにおいて、3ビットの増加の結果である4つの“0”ビットが平均して現れる(直接ビットコード化法の適用により得られた4ビットから、最後のバイト上の分析ビットを除去することにより節約された1ビットを引く)。しかし、作業領域をを1バイトから2バイトに変え、図38に示す方法でコード化することにより、“0”ビットの値が発生する可能性は1/16, 777, 216となる。更に4バイトの作業領域が選択されると、“0”ビットの値が発生する可能性は1/4, 294, 967, 296となる。従って、十分に大きい作業領域を選択することにより、ワード値“0”が発生する可能性は非常に小さい。ワード値“0”が発生する場合においてさえ、その再帰での縮小は再帰の連続により更に拡大する。

この発明による圧縮法の好適実施例について説明してきたが、この発明の精神から離れることなく、この発明を説明するために選択された好適実施例に種々の変形又は追加を行ってもよい。例えば、圧縮法を説明するために、ここで選択さ

れた多数の種々のものが特定の長さを持って選択されている（1ビット、1ニブル、1ワード等のような）。多くの例においてこれらのワードのサイズは柔軟性があり、種々のアルゴリズムの処理に影響することなく、容易に変更できる。従って、保護を受けようとする主題は、添付のクレームにおいて定義されている内容を拡張し、それについての全ての等価物を含む。

【図1】

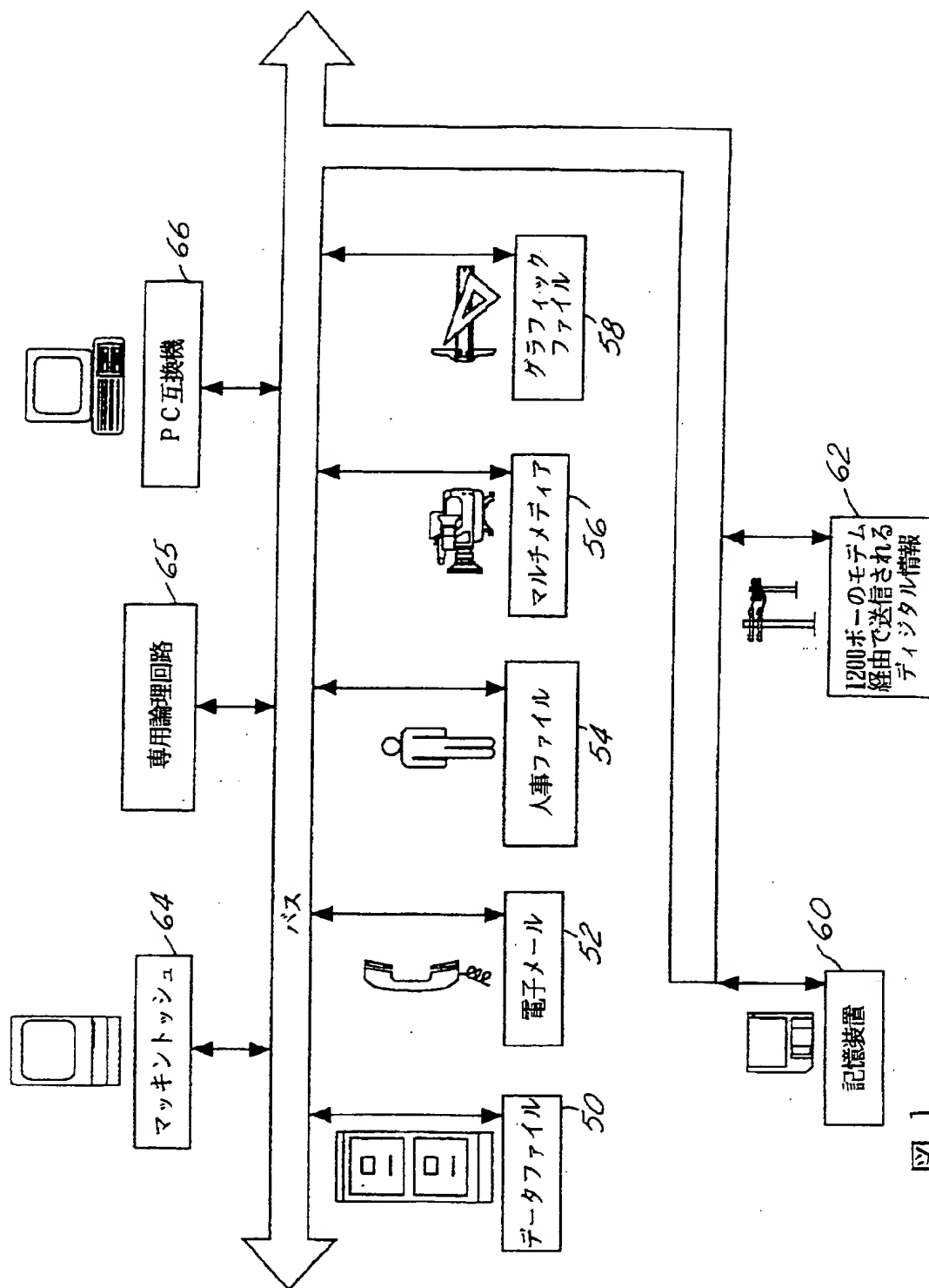


図1

【図2】

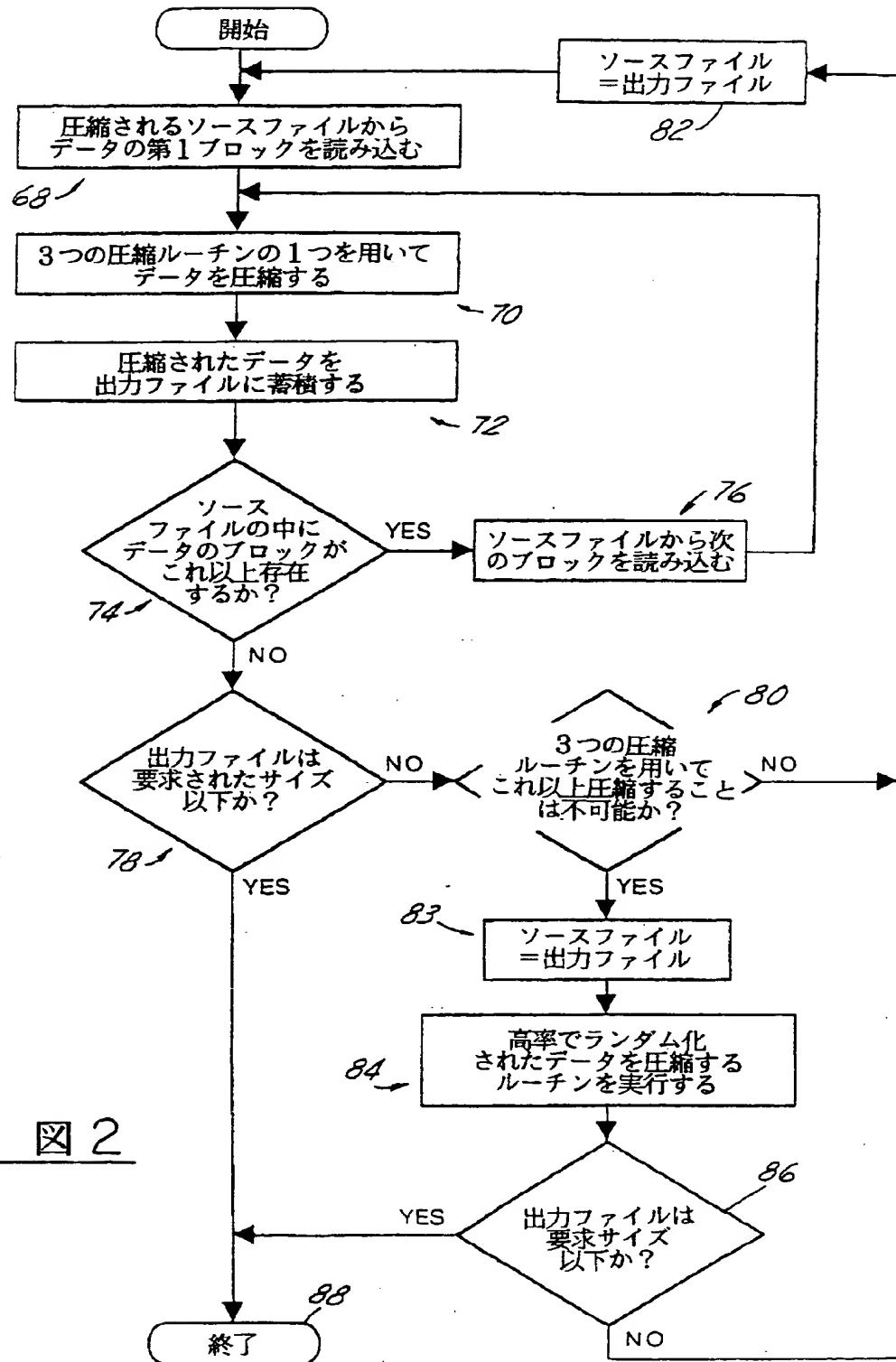
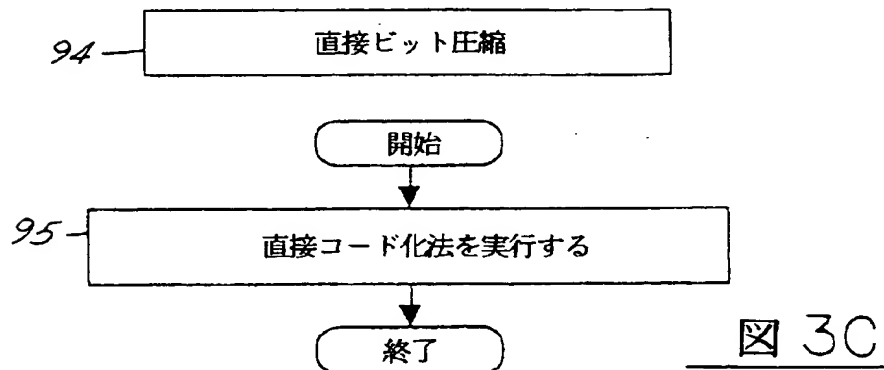
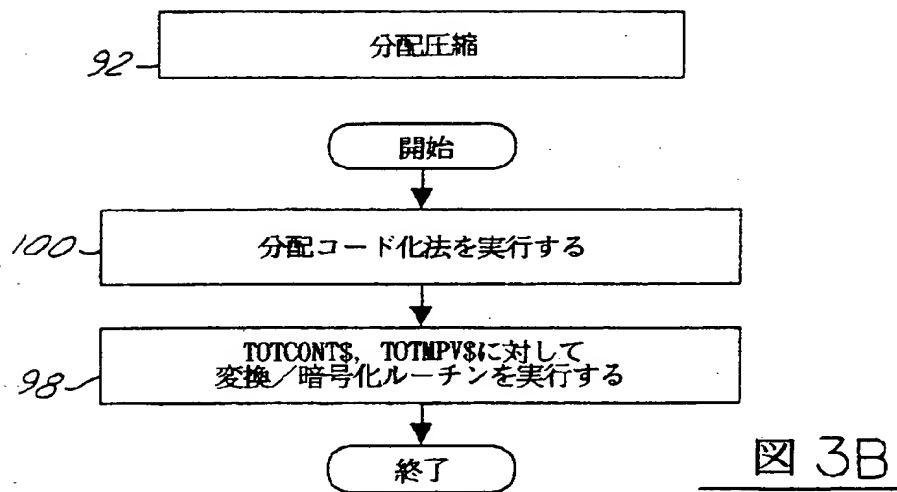
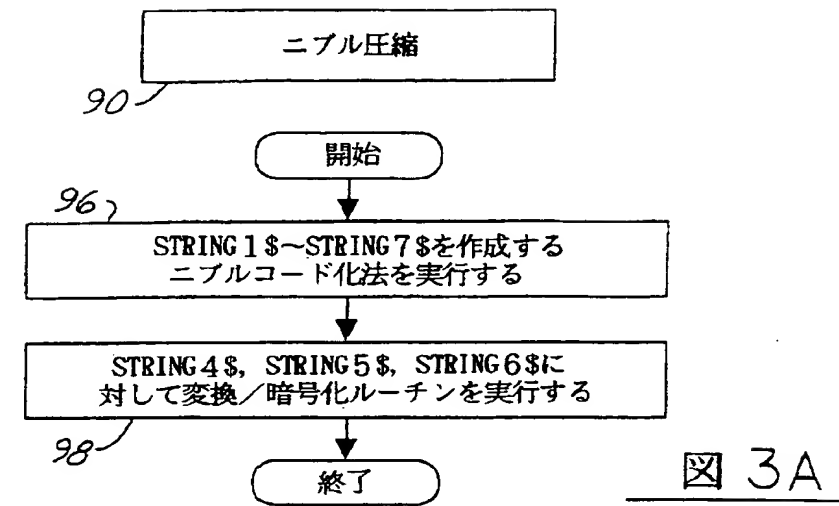


図 2

【図3】



【図4】

ニブルコード化法							
任意の ニブルの値 列	S T R I N G 1 \$	S T R I N G 2 \$	S T R I N G 3 \$	S T R I N G 4 \$	S T R I N G 5 \$	S T R I N G 6 \$	S T R I N G 7 \$
0000	0	0	00	X	X	X	X
0001	0	0	10	X	X	X	X
0010	0	0	01	X	X	X	X
0011	0	0	11	X	X	X	X
0100	0	1	00	X	X	X	X
0101	0	1	10	X	X	X	X
0110	0	1	01	X	X	X	X
0111	0	1	11	X	X	X	X
1000	1	X	00	0	0	X	X
1001	1	X	10	0	0	X	X
1010	1	X	01	0	0	X	X
1011	1	X	11	0	0	X	X
1100	1	X	X	0	1	X	X
1101	1	X	X	1	X	1	X
1110	1	X	X	1	X	0	0
1111	1	X	X	1	X	0	1

X = 変化無し

図 4

【図5】

出力列	バイト1		バイト2		バイト3		バイト4		バイト5	
	ニブル1	ニブル2	ニブル3	ニブル4	ニブル5	ニブル6	ニブル7	ニブル8	ニブル9	ニブル10
	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001
STRING1\$	0	0	0	0	0	0	0	0	1	1
STRING2\$	0	0	0	0	1	1	1	1	X	X
STRING3\$	00	10	01	11	00	10	01	11	00	10
STRING4\$	X	X	X	X	X	X	X	X	0	0
STRING5\$	X	X	X	X	X	X	X	X	0	0
STRING6\$	X	X	X	X	X	X	X	X	X	X
STRING7\$	X	X	X	X	X	X	X	X	X	X

X = 変化無し

図5A

バイト6		バイト7		バイト8		総ビット数		0の総ビット数 に対する比率
ニブル11	ニブル12	ニブル13	ニブル14	ニブル15	ニブル16			
1010	1011	1100	1101	1110	1111	64		50 %
1	1	1	1	1	1	16		50.0 %
X	X	X	X	X	X	8		50.0 %
01	11	X	X	X	X	24		50.0 %
0	0	0	1	1	1	8		62.5 %
0	0	1	X	X	X	5		80.0 %
X	X	X	1	0	0	3		66.6 %
X	X	X	X	0	1	2		50.0 %
						66	64	

図5B

【図6】

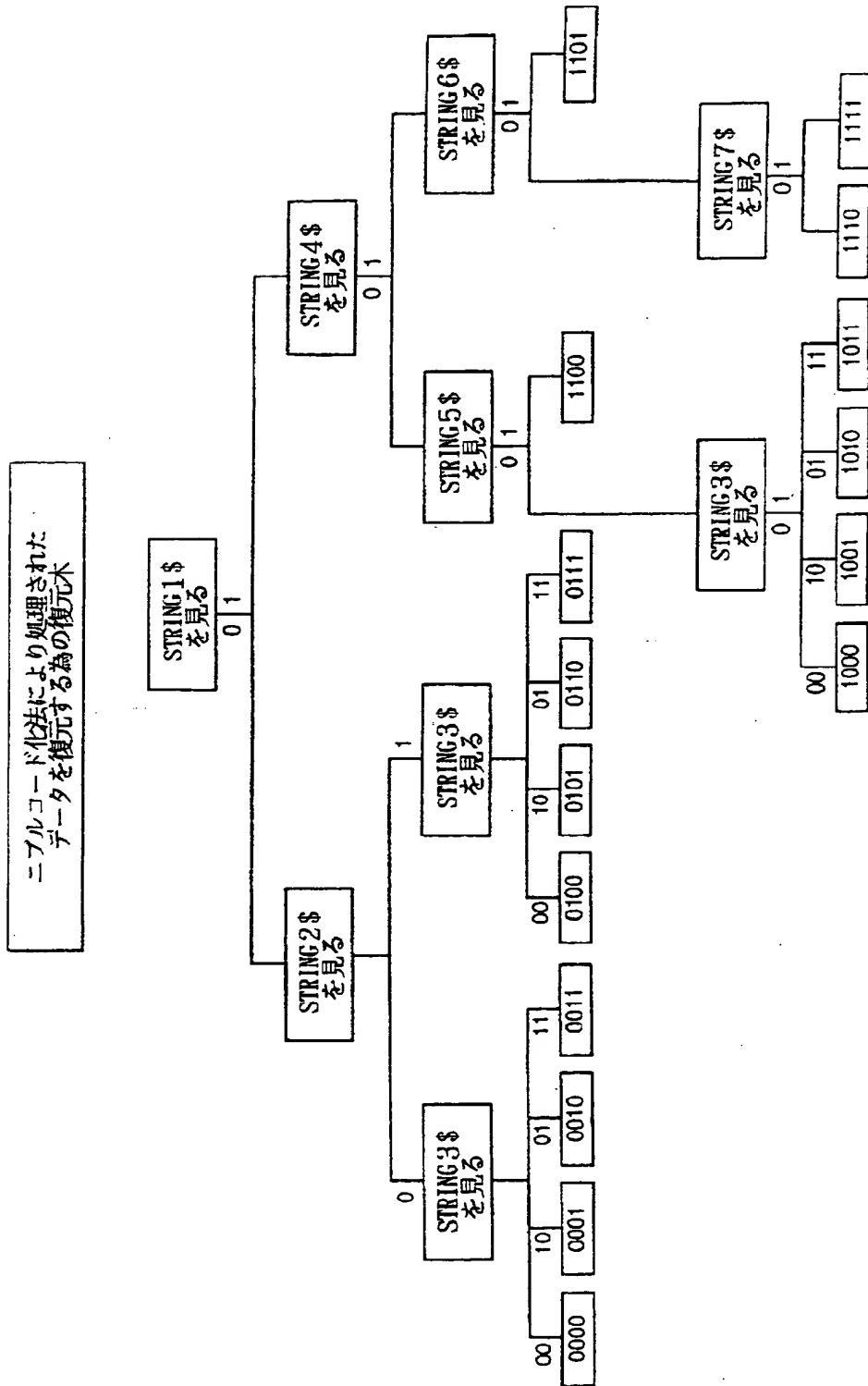


図6

【図 7】

図 4 に示すニブルコード化法の使用によるコード化					
出力列	入力列	ニブル 1	ニブル 2	ニブル 3	ニブル 4
		0001	0010	0011	1110
	STRING1\$	0	0	0	1
	STRING2\$	0	0	0	X
	STRING3\$	10	01	11	X
	STRING4\$	X	X	X	1
	STRING5\$	X	X	X	X
	STRING6\$	X	X	X	0
	STRING7\$	X	X	X	0

X = 変化無し

図 7A

図 6 に示す復元木の使用によるニブル 1 の復元	
出力列	ビット配置 1 2 3 4 5 6 7 8 9 10
STRING1\$	0 0 0 1
STRING2\$	0 0 0
STRING3\$	1 0 0 1 1 1
STRING4\$	1
STRING5\$	
STRING6\$	0
STRING7\$	0

入力列 = 0001

図 7B

図 6 に示す復元木の使用によるニブル 2 の復元	
出力列	ビット配置 1 2 3 4 5 6 7 8 9 10
STRING1\$	0 0 1
STRING2\$	0 0
STRING3\$	0 1 1 1
STRING4\$	1
STRING5\$	
STRING6\$	0
STRING7\$	0

入力列 = 00010010

図 7C

【図 7】

図 6 に示す復元木の使用によるニブル 3 の復元										
出力列	ビット配置 1 2 3 4 5 6 7 8 9 10									
STRING1\$	0	1								
STRING2\$	0									
STRING3\$	1	1								
STRING4\$	1									
STRING5\$										
STRING6\$	0									
STRING7\$	0									
入力列 = 000100100011										

図 7D

図 6 に示す復元木の使用によるニブル 4 の復元										
出力列	ビット配置 1 2 3 4 5 6 7 8 9 10									
STRING1\$	1									
STRING2\$										
STRING3\$										
STRING4\$	1									
STRING5\$										
STRING6\$	0									
STRING7\$	0									
入力列 = 0001001000111110										

図 7E

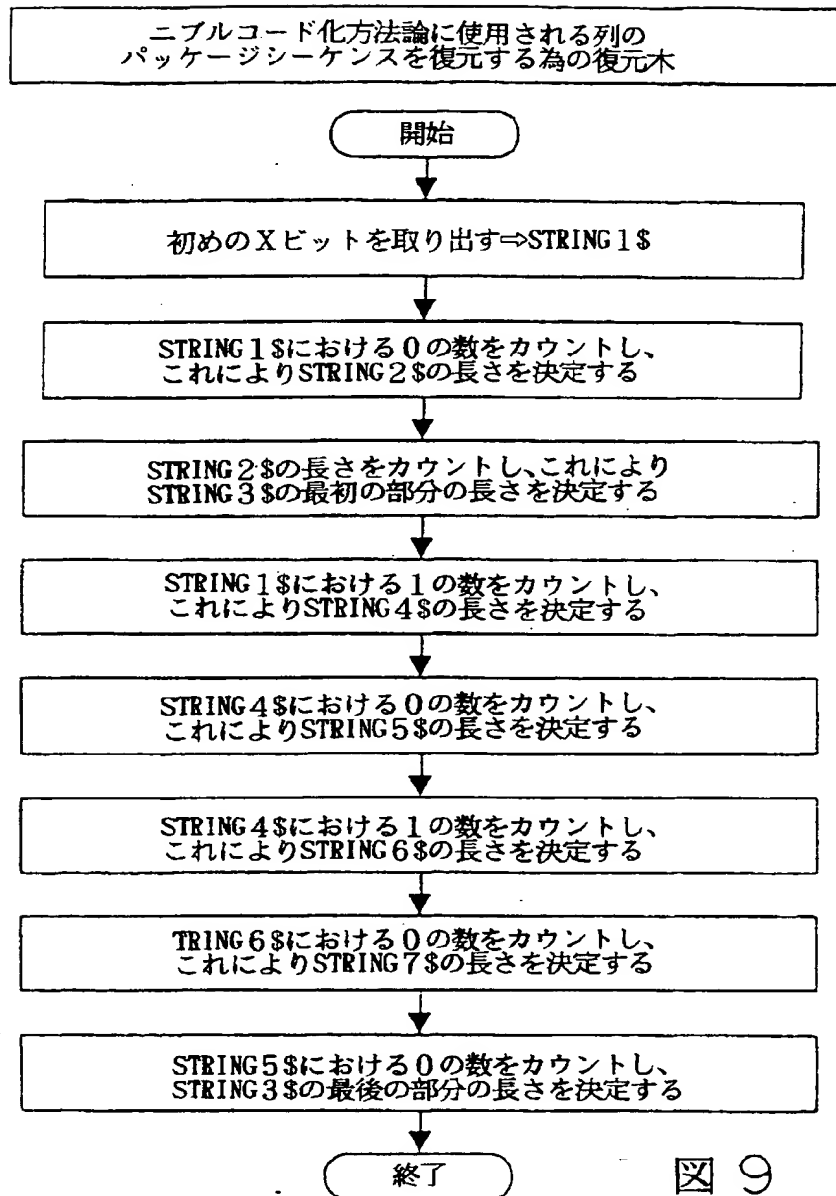
【図 8】

ニブルコード化方法論に使用される
列のパッケージシーケンス

{ STRING1\$ / STRING2\$ / STRING3\$ (最初の部分) /
STRING4\$ / STRING5\$ / STRING6\$ / STRING7\$ / STRING3\$ (最後の部分) }

図 8

【図9】



【図10】

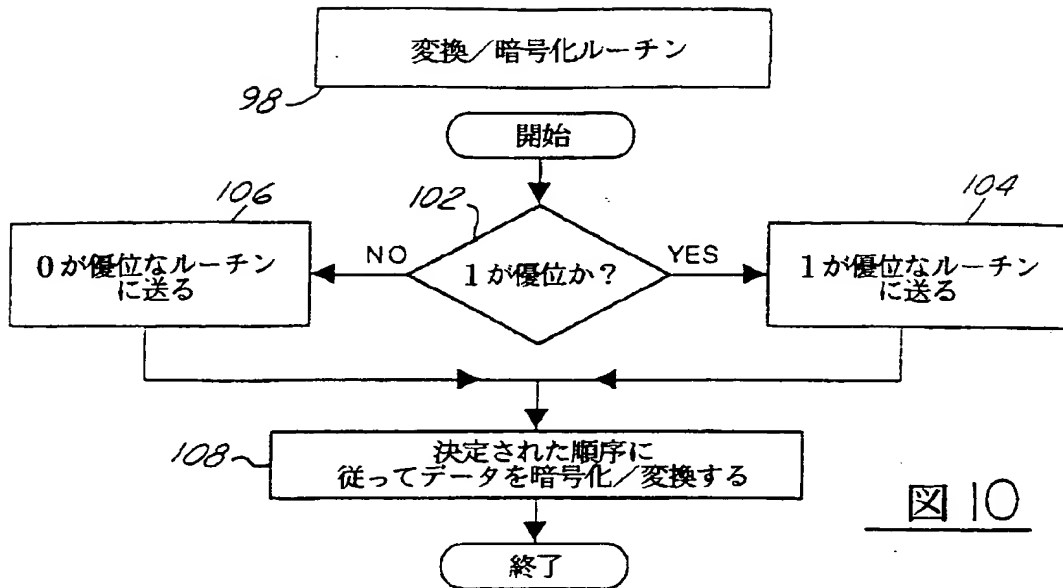


図 10

【図11】

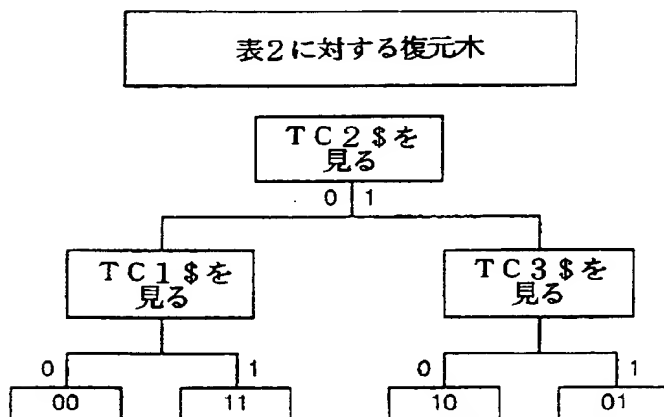
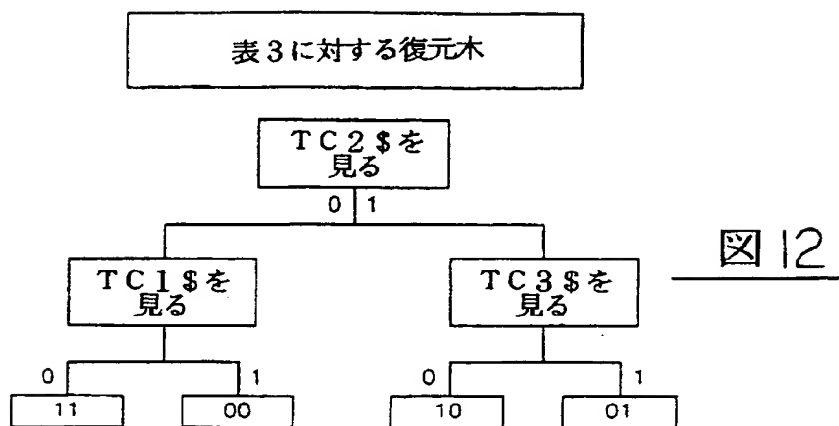


図 11

【図12】



【図13】

表2に示す変換方法論の使用による変換	
入力列	0010100010・0010000010
出力列	
TC1\$	0 X X 0 X 0 X 0 0 X
TC2\$	0 1 1 0 1 0 1 0 0 1
TC3\$	X 0 0 X 0 X 0 X X 0
X = 変化無し	

図13A

図11に示す復元木の使用による復元	
出力列	ビット配置 1 2 3 4 5 6 7 8 9 10
TC1\$	00000
TC2\$	0110101001
TC3\$	0000000000
入力列 = 00	

図13B

図11に示す復元木の使用による復元	
出力列	ビット配置 1 2 3 4 5 6 7 8 9
TC1\$	0000
TC2\$	0110101001
TC3\$	00000
入力列 = 0010	

図13C

図11に示す復元木の使用による復元	
出力列	ビット配置 1 2 3 4 5 6 7 8 9 10
TC1\$	0000
TC2\$	0110101001
TC3\$	0000
入力列 = 001010	

図13D

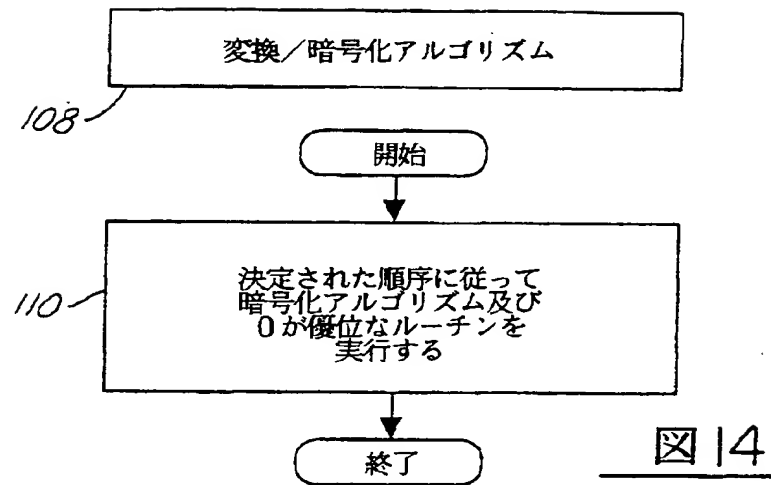
図11に示す復元木の使用による復元	
出力列	ビット配置 1 2 3 4 5 6 7 8 9 10
TC1\$	0000
TC2\$	0110101001
TC3\$	000
入力列 = 00101000	

図13E

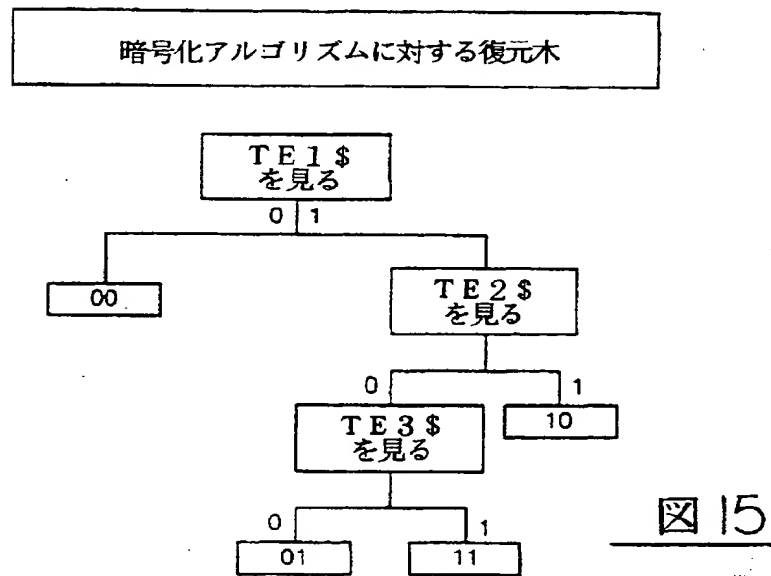
図11に示す復元木の使用による復元	
出力列	ビット配置 1 2 3 4 5 6 7 8 9 10
TC1\$	000
TC2\$	01101001
TC3\$	000
入力列 = 0010100010	

図13F

【図14】



【図15】



【図 16】

表 4 に示す暗号化ルーチンの使用による暗号化											
出力列 \ 入力列			ビット配置								
			1	2	3	4	5	6	7	8	9
			0 0 1 0 0 0 1 0 0 0								
TE1\$			0	1	0	1	0				
TE2\$			X	1	X	1	X				
TE3\$			X	X	X	X	X				
X = 変化無し											

図 16A

図 15 に示す復元木の使用による復元										
出力列		ビット配置								
		1	2	3	4	5	6	7	8	9
TE1\$		0	1	0	1	0				
TE2\$		1	1							
TE3\$										
入力列 = 00										

図 16B

図 15 に示す復元木の使用による復元										
出力列		ビット配置								
		1	2	3	4	5	6	7	8	9
TE1\$		0	1	0	1	0				
TE2\$		1	1							
TE3\$										
入力列 = 0010										

図 16C

図 15 に示す復元木の使用による復元										
出力列		ビット配置								
		1	2	3	4	5	6	7	8	9
TE1\$		0	1	0						
TE2\$		1								
TE3\$										
入力列 = 001000										

図 16D

図 15 に示す復元木の使用による復元										
出力列		ビット配置								
		1	2	3	4	5	6	7	8	9
TE1\$		0	1	0						
TE2\$		1								
TE3\$										
入力列 = 00100010										

図 16E

図 15 に示す復元木の使用による復元										
出力列		ビット配置								
		1	2	3	4	5	6	7	8	9
TE1\$		0								
TE2\$										
TE3\$										
入力列 = 0010001000										

図 16F

【図 17】

0 が優先なルーチン													
入力列 STRING4\$		4	8	12	16	20	24	28	32	36	40		
出力列		0010	1000	1000	1000	0010	0000	0001	1010	0000	0010		
TC1\$		0	X	0	X	0	0	0	X	X	0	0	X
TC2\$		0	1	0	1	0	0	1	0	1	0	0	1
TC3\$		X	0	0	X	0	X	X	1	0	0	X	0

64ビット-64ビット=変換法により除去されるのは0ビット

X = 変化無し

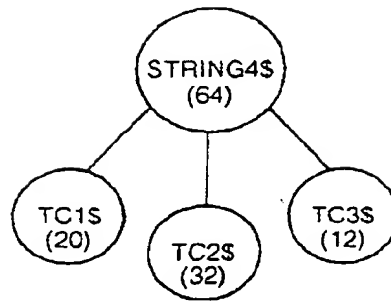
図 17A

0の総ビット数 に対する比率 (すなわち0のバイアス)		列における 総ビット数	
50/64 ⇒ .78125		← 64 ビット	
19/20 ⇒ .95000		← 20 ビット	
20/32 ⇒ .62500		← 32 ビット	
9/12 ⇒ .75000		← 12 ビット	
		64 ビット	

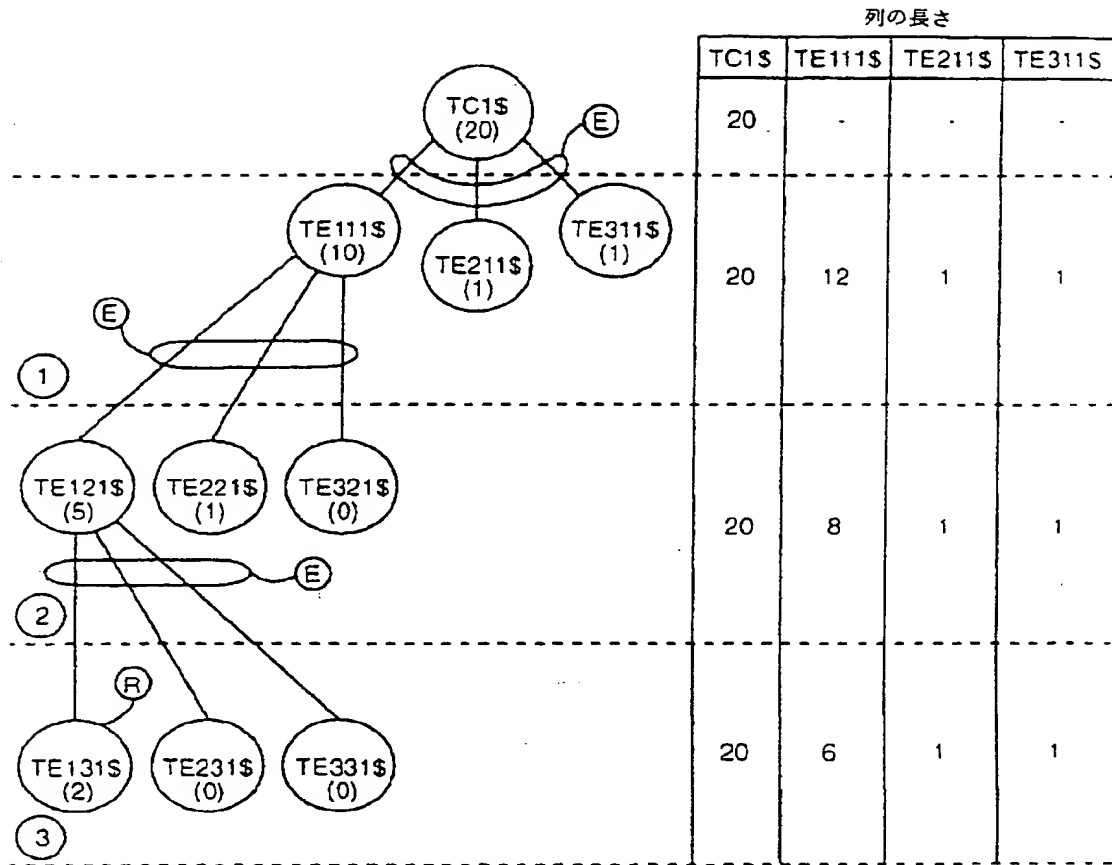
44	48	52	56	60	64
0000	0000	1000	0011	0000	0101
0	0	0	0	0	0
0	0	0	0	0	0
X	X	X	X	X	1

図 17B

【図18】

図 18

【図 19】



TC1\$ の圧縮された長さ = $6+1+1$
 = 8 ビット

総縮小 = $20-8$
 = 12 ビット

縮小率 = $12/20$
 = 60%

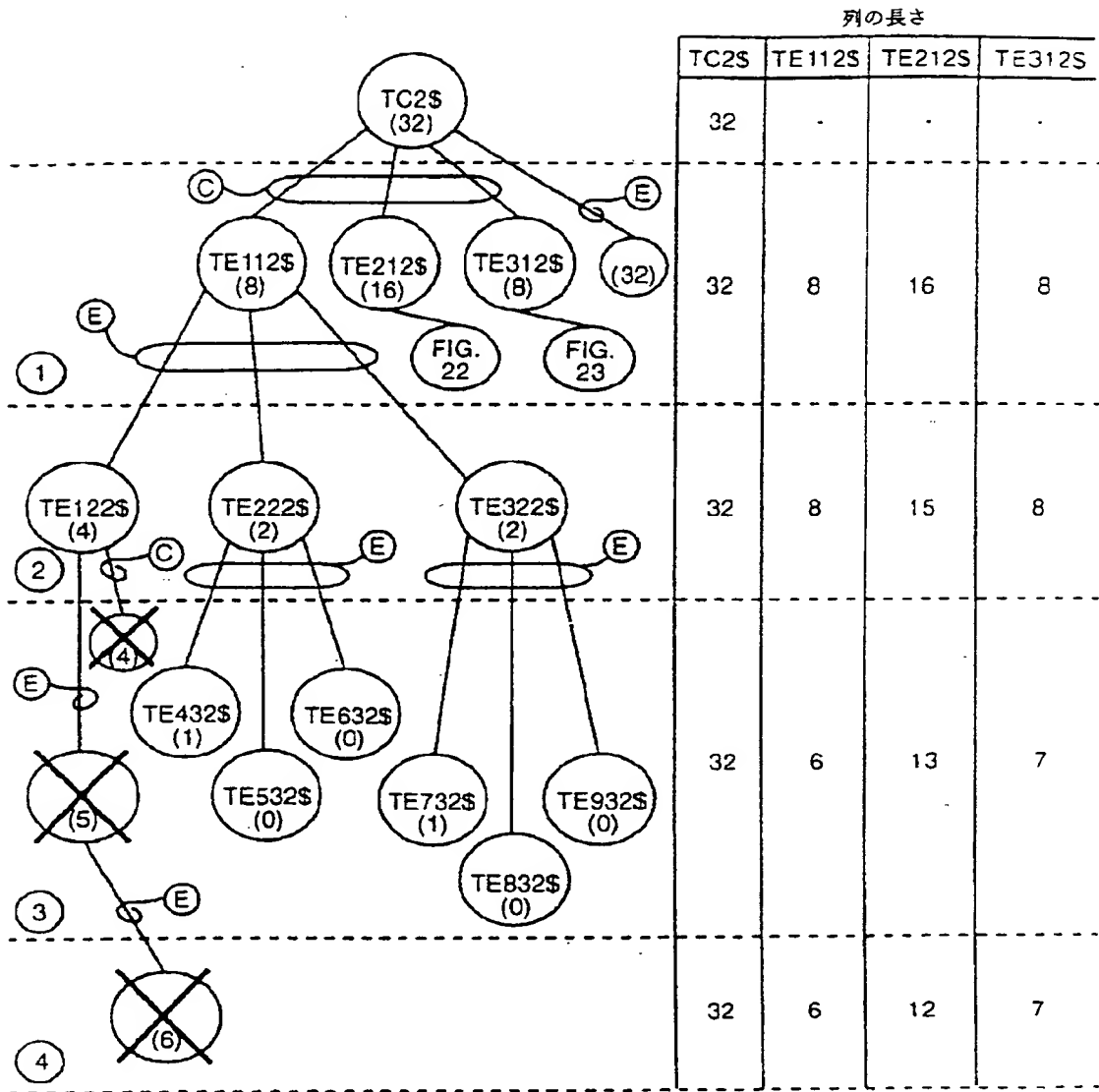
図 19

20

[illegible]

節約できるビット数の合計 $= 8 + 4 + 2 = 14$ ビット

【図 21】



最終的なTC2\$の長さ = 6+12+7
= 25 ビット

総縮小 = 32-25
= 7 ビット

縮小率 = 7/32
= 21.875%

列の長さ

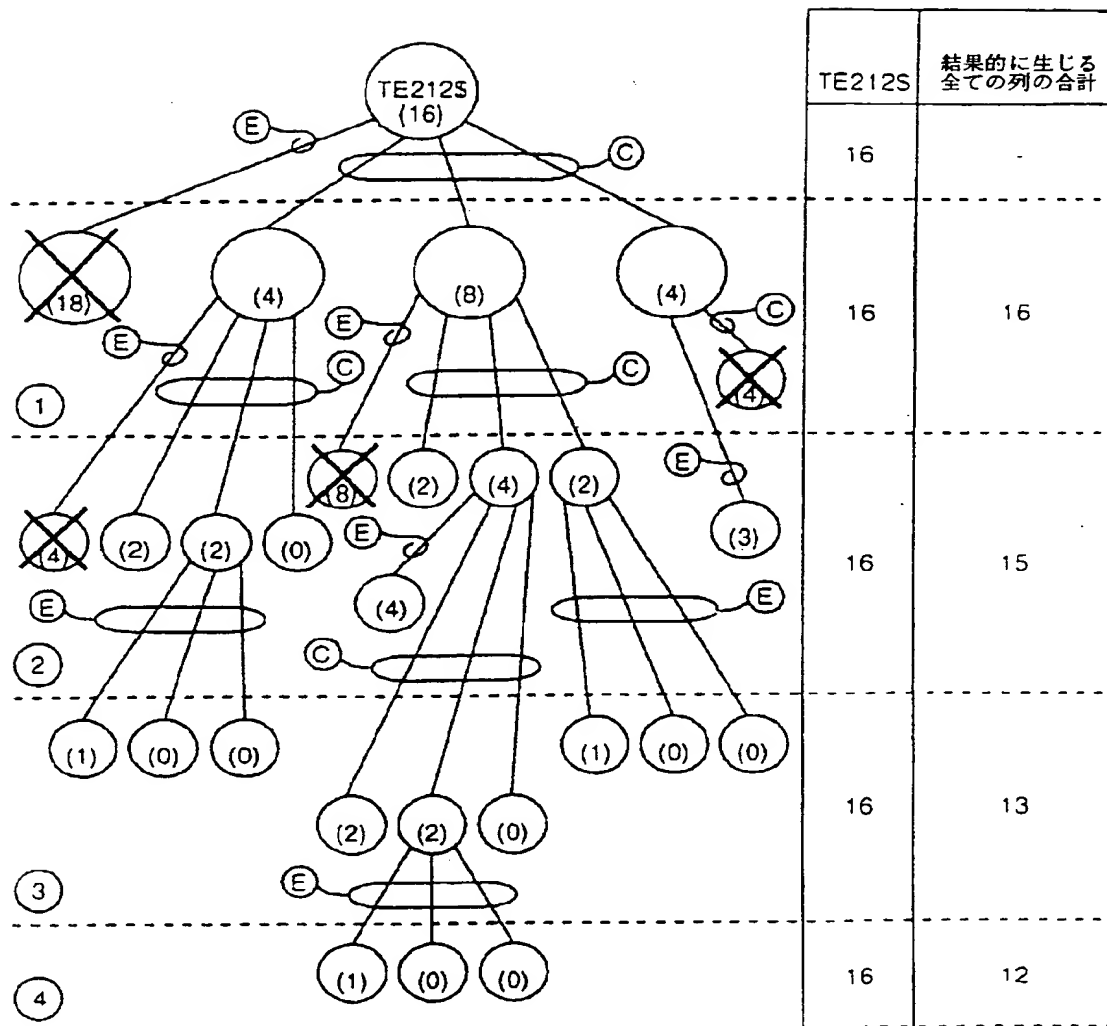


图 22

【図 23】

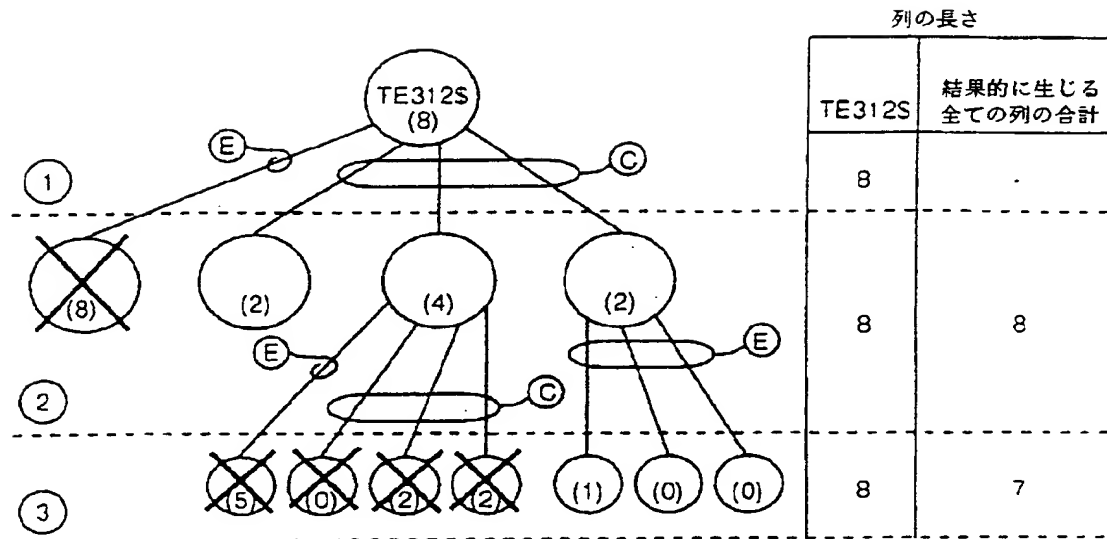
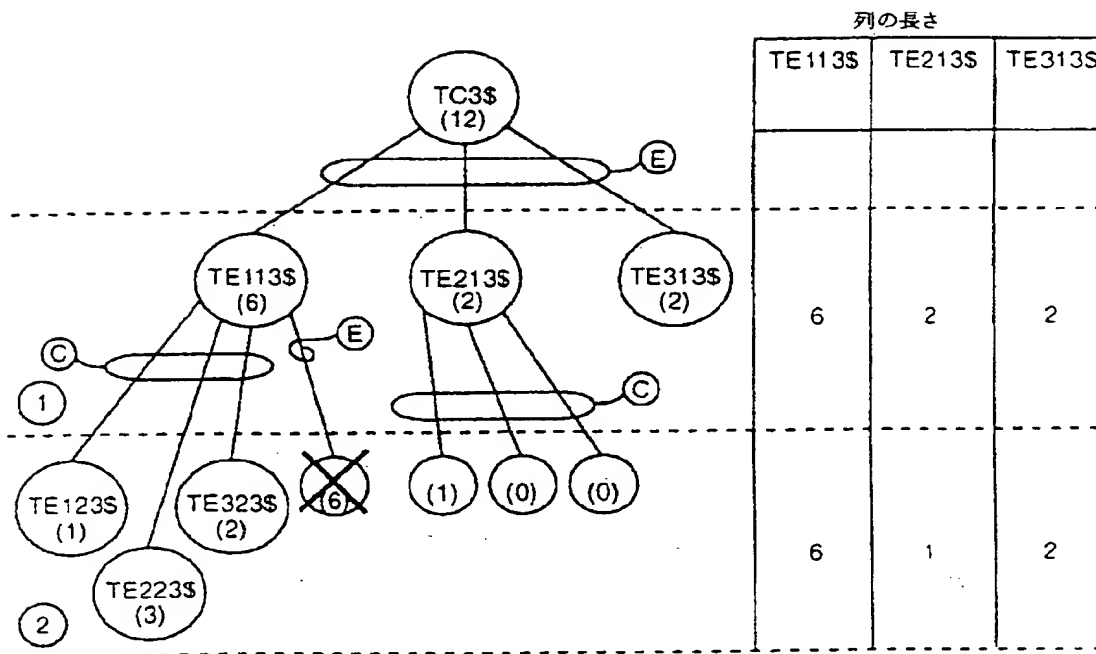


図 23

【図 24】



最終的なTC3\$の長さ = 6+1+2
= 9ビット

総縮小 = 12-9
= 3ビット

縮小率 = 3/12
= 25%

図 24

【図 2 5】

単一の暗号化の例				
		暗号化	総ビット数	1の総数
入力列	出力列	11001010001101100101001101000000	32	13
TE1\$	1 0 1 1 0 1 1 1 1 1 0 1 1 0 0 0	16	10	6
TE2\$	0 X 1 1 X 0 0 1 0 0 X 0 0 X X X	10	3	7
TE3\$	1 X X X X 1 0 X 0 0 X 1 0 X X X	7	3	4
X = 変化無し				
TE1\$ / TE2\$ / TE3\$				
= 101101111011000/0110010000/1100010				

図25

【図26】

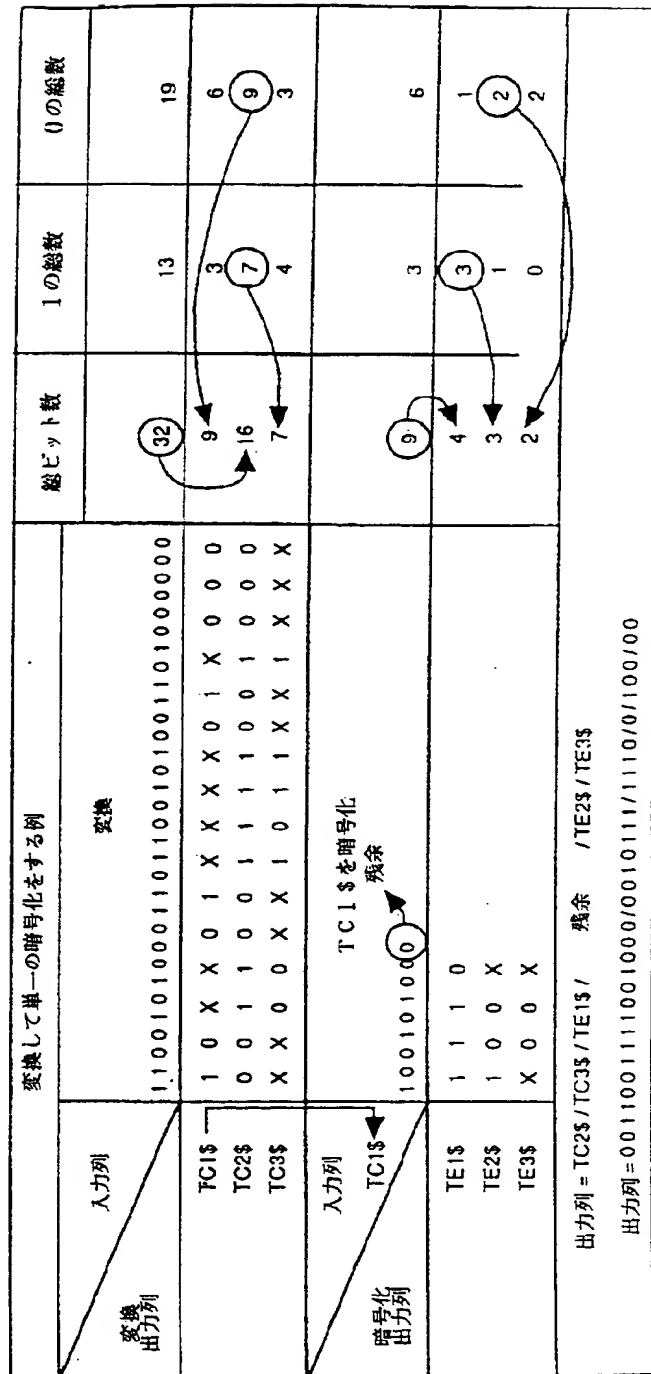


図26

27A

[illegible]

【図27】

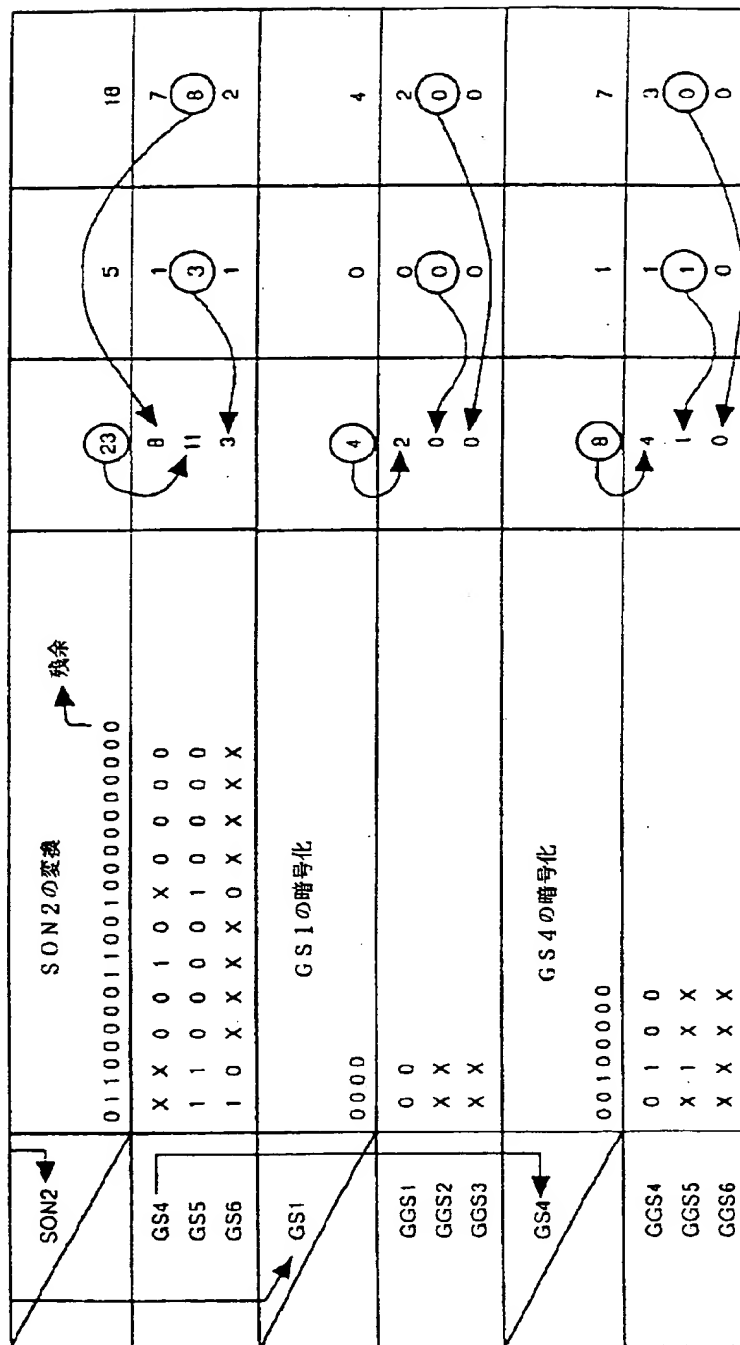


図 27C

【图 28】

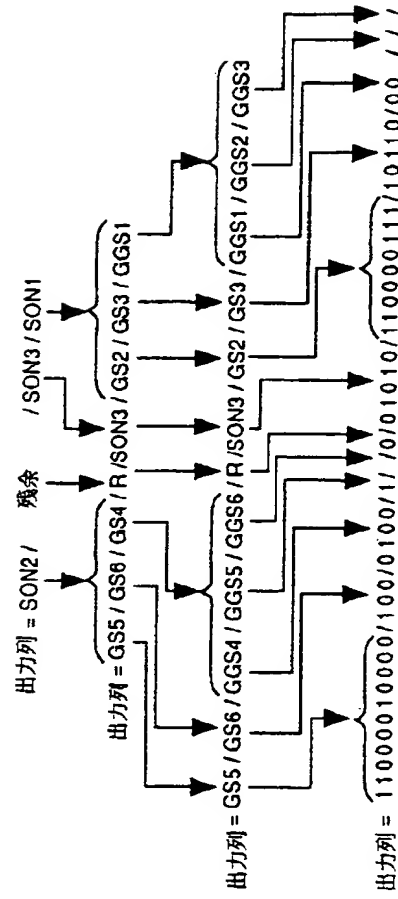


图 28

【図29】

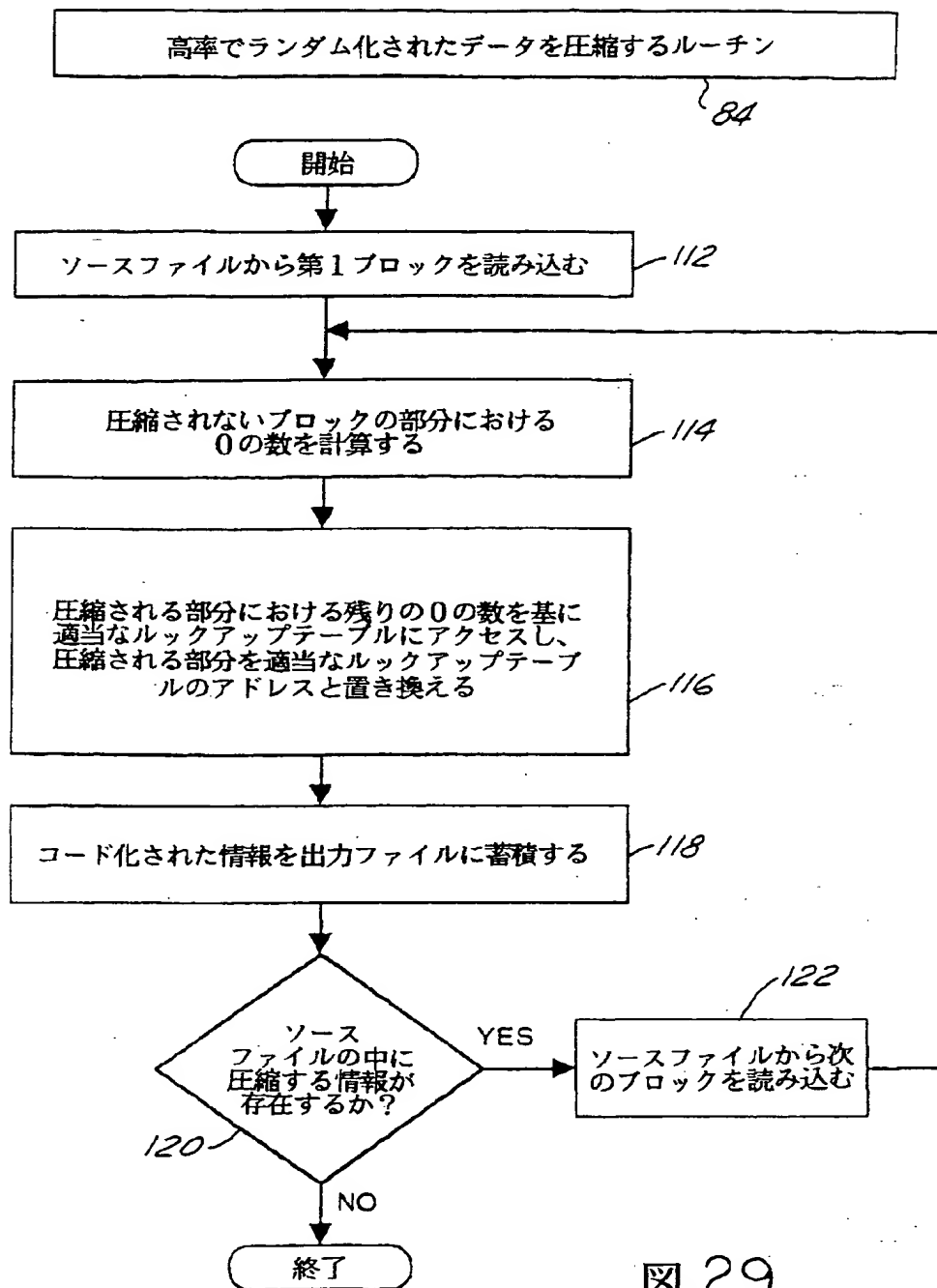


図 29

【図30】

	ルックアップ テーブル#1	0が9ケのビット パターン
1	0000	0000000001
2	0001	0000000010
3	0010	0000000100
4	0011	0000001000
5	0100	0000010000
8	1000	0010000000
9	1001	0100000000
10	1010	1000000000

図 30A

	ルックアップ テーブル#2	0が8ケのビット パターン
1	000000	0000000011
2	000001	0000000101
3	000010	0000000110
4	000011	0000001010
5	000100	0000011000
43	101011	0110000000
44	101100	1010000000
45	101101	1100000000

図 30B

【図 31】

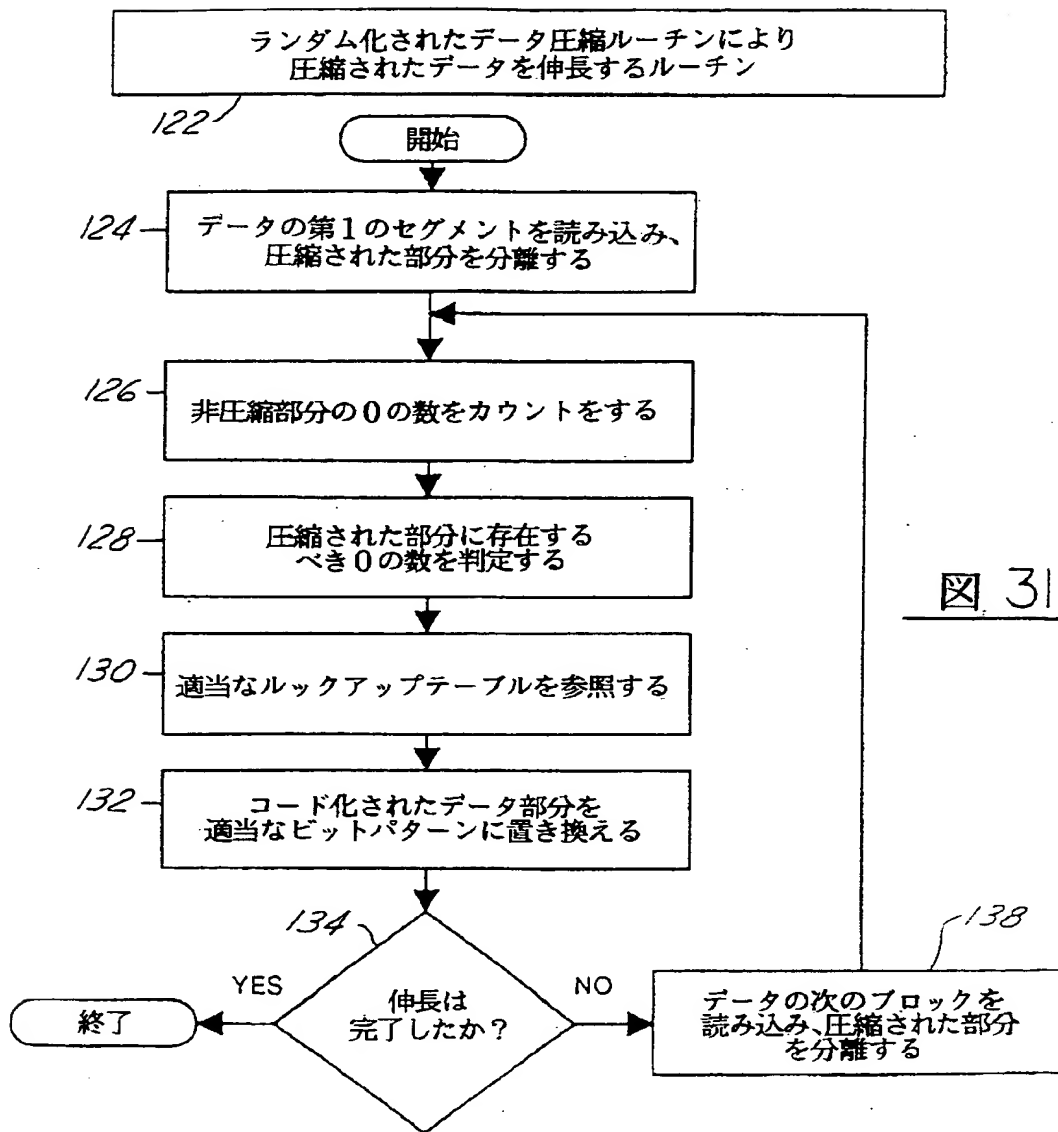


図 31

【図32】

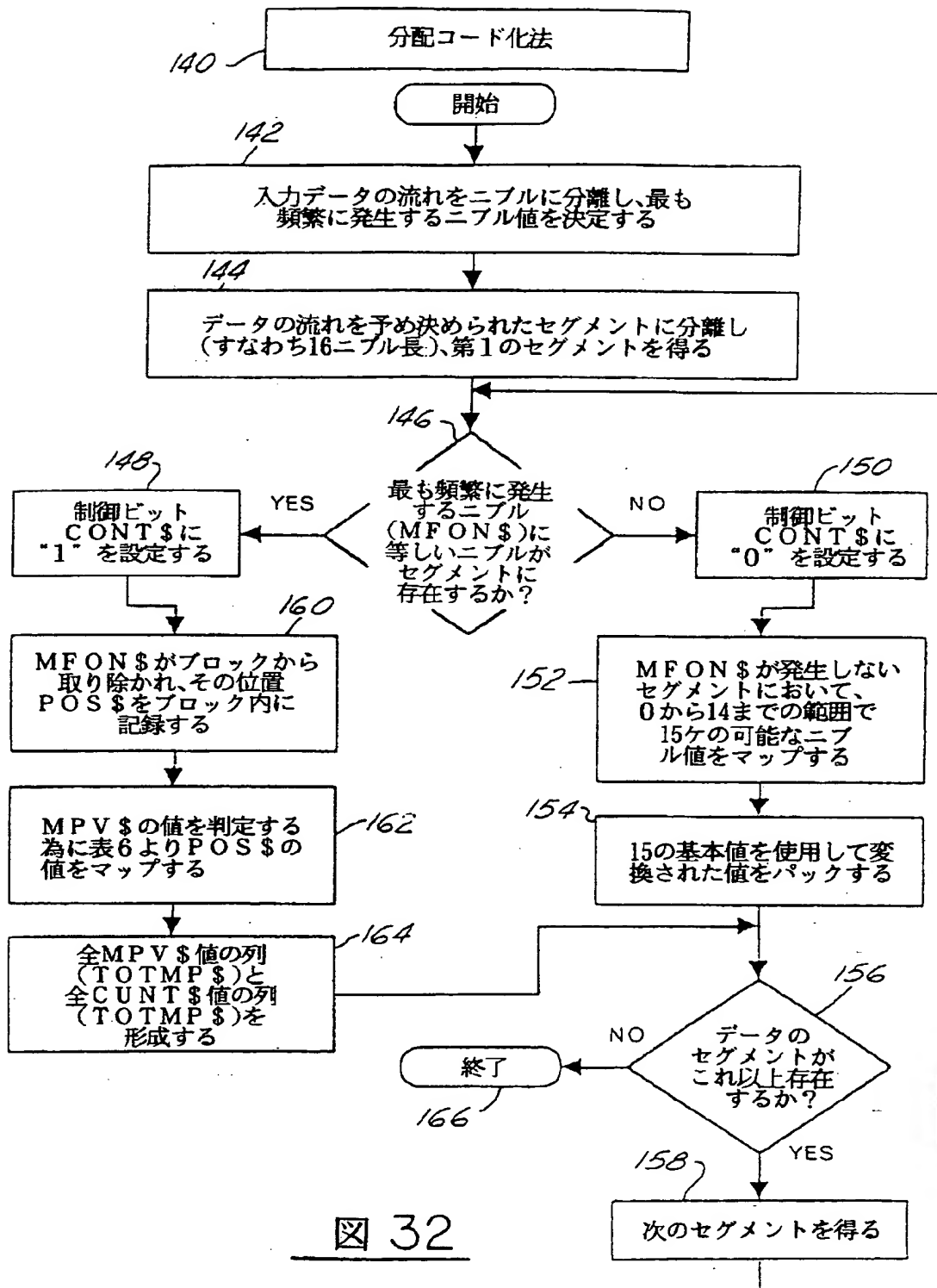


図 32

【図 3 3】

	ニブル 1	ニブル 2	ニブル 3	ニブル 4	ニブル 5	ニブル 16	CONT\$	POS\$
セグメント 1	1101	1010	1010	0110	1000	0010	0	XXXX
セグメント 2	0010	1110	1101	1001	0010	1110	1	0010
セグメント 3	0110	0010	0000	1011	1110	0111	1	0101
・	・	・	・	・	・	・	・	・
・	・	・	・	・	・	・	・	・
・	・	・	・	・	・	・	・	・
セグメント 99	1000	1110	0011	1100	1000	1100	0	XXXX
セグメント 100	0111	1111	1011	0000	100	1110	1	1111

図 33A

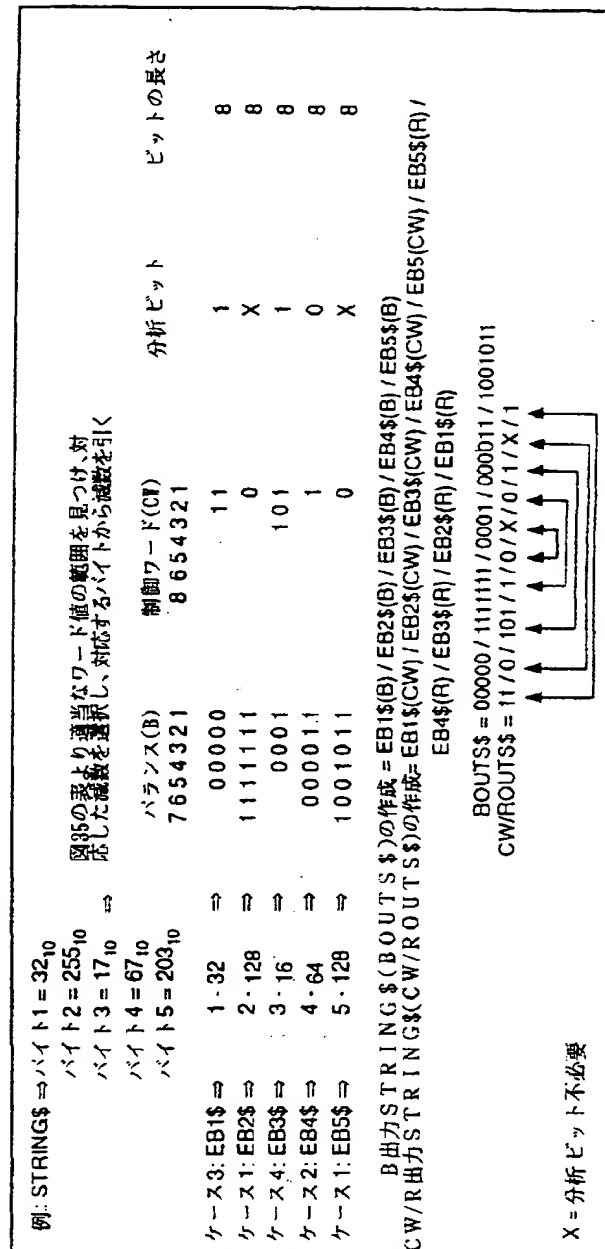
	ニブル 1	ニブル 2	ニブル 3	ニブル 4	ニブル 5	ニブル 16	CONT\$	POS\$	MPV\$
セグメント 1	1101	1010	1010	0010	1000	0010	0	XXXX	XXXX
セグメント 2	0010	1101	1001	0010	0010	XXXX	1	0010	0010
セグメント 3	0110	0010	0000	1011	1000	XXXX	1	0101	1000
・	・	・	・	・	・	・	・	・	・
・	・	・	・	・	・	・	・	・	・
・	・	・	・	・	・	・	・	・	・
セグメント 99	1000	1110	0011	1100	100	1100	0	XXXX	XXXX
セグメント 100	0111	1111	1011	0000	10	XXXX	1	1111	1111

図 33B

【図34】

$$\boxed{\text{図34}} \left\{ \begin{array}{l} \text{出力列} = \text{MOD}(\text{TOTCONT\$}) / \text{MOD}(\text{TOTMVP\$}) / \text{パックされたブロック1/ブロック2の残りの部分、ブロック2の残りの部分、} \\ \text{ブロック3のバックされた部分、ブロック3の残りの部分} / \dots \\ \text{パックされたブロック99/ブロック100のバックされた部分、ブロック100の残りの部分} \end{array} \right.$$

直接ビットコード化(1バイトワード)								
ケース	ワード値の範囲	減数	バランス	コード化する 必要ビット数	制御ワード	コード化された ビットのワード	分析ビット	コード化された ビット+制御ワード +分析ビット
1	256 > > 127	$2^8 \div 2 = 128$	0 - 127	7	0	8	X	8
2	128 > > 63	$2^8 \div 4 = 64$	0 - 63	6	1	7	0	8
3	64 > > 31	$2^8 \div 8 = 32$	0 - 31	5	11	7	1	8
4	32 > > 15	$2^8 \div 16 = 16$	0 - 15	4	101	7	1	8
5	16 > > 7	$2^8 \div 32 = 8$	0 - 7	3	1001	7	1	8
6	8 > > 3	$2^8 \div 64 = 4$	0 - 3	2	10001	7	1	8
7	4 > > 1	$2^8 \div 128 = 2$	0 - 1	1	100001	7	1	8
8	2 > > 0	$2^8 \div 256 = 1$	0	0	1000001	7	1	8
9	0	0	0	0	10000001	8	1	9
X = 不必要								



【図37】

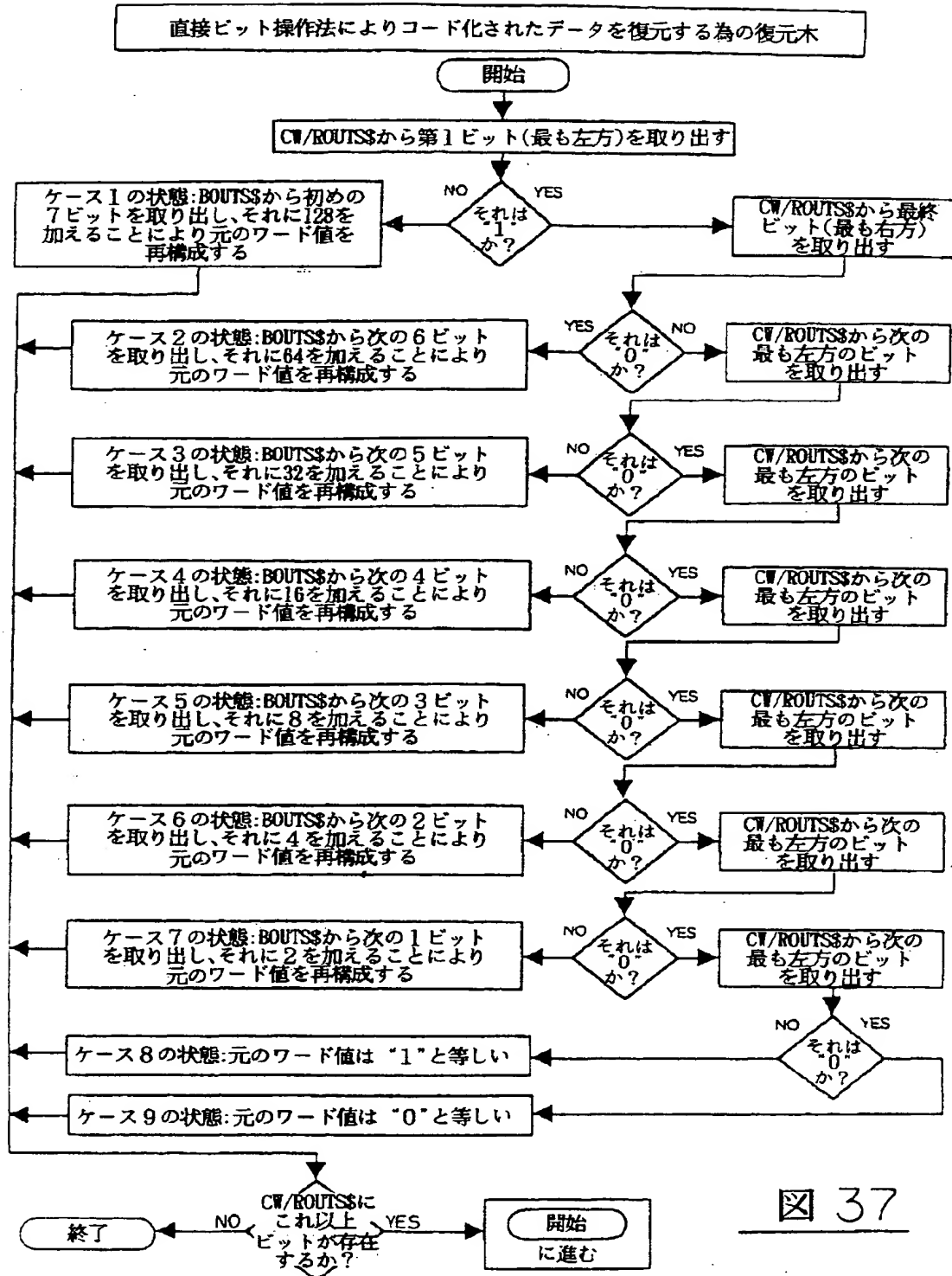


図 37

【図 3 8】

直接ビットコード化法によって再帰されるデータ列の作成

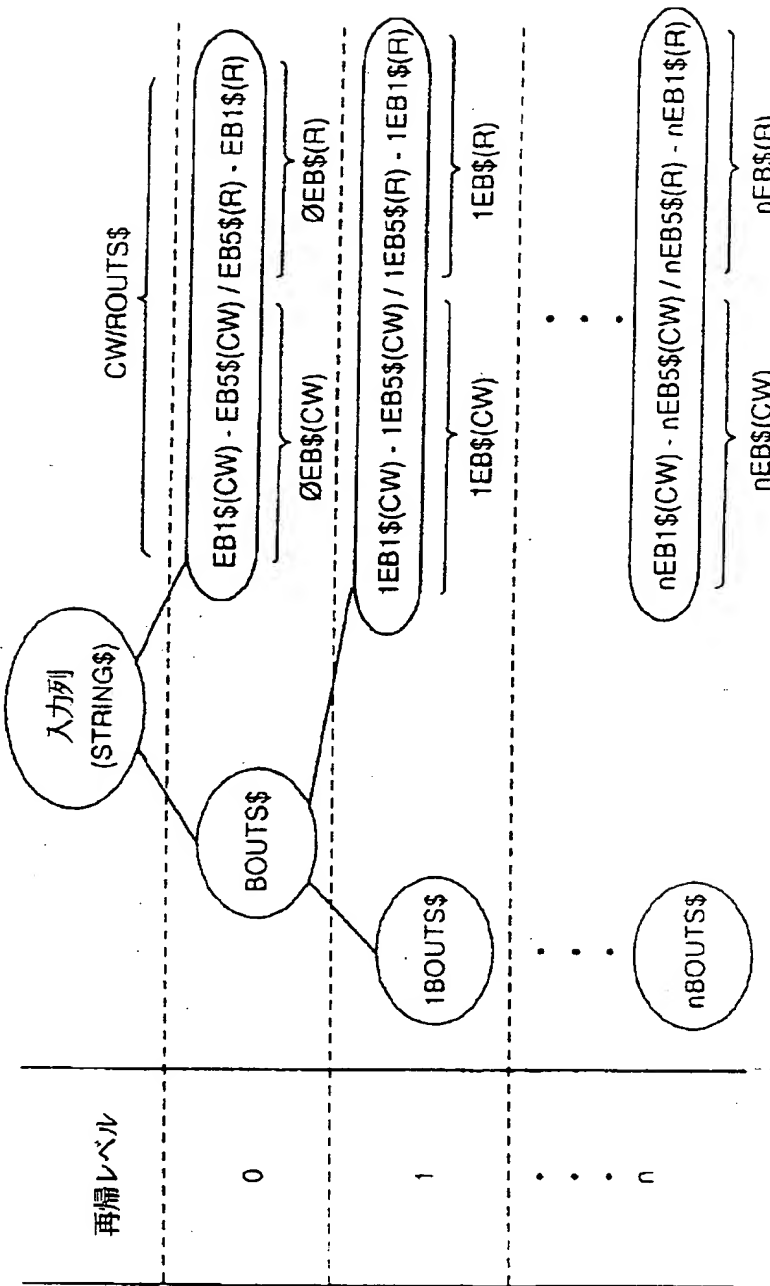


図 38A

【図38】

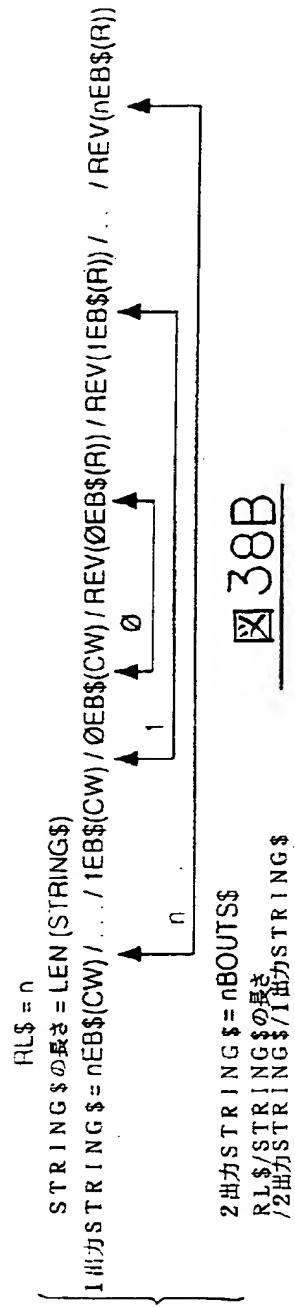


図38B

図38C

【図39】


直接ビットコード化(2バイトワード)								
ケース	ワード値の範囲	減数	バランス	バランス値を コード化する為の 必要ビット数	制御ワード	コード化 された ビット+ 制御ワードの ビット	分析ビット	コード化 された ビット+ 制御ワード+ 分析ビット
1	$2^{16} \div 1 > > (2^{16} \div 2) - 1$	$2^{16} \div 2$	$0 - (2^{16} \div 2) - 1$	15	0	16	X	16
2	$2^{16} \div 2 > > (2^{16} \div 4) - 1$	$2^{16} \div 4$	$0 - (2^{16} \div 4) - 1$	14	1	15	0	16
3	$2^{16} \div 4 > > (2^{16} \div 8) - 1$	$2^{16} \div 8$	$0 - (2^{16} \div 8) - 1$	13	11	15	1	16
.
.
.
15	$2^{16} \div 2^{14} > > (2^{16} \div 2^{15}) - 1$	$2^{16} \div 2^{15}$	$0 - (2^{16} \div 2^{15}) - 1$	1	1000000000000001	15	1	16
16	$2^{16} \div 2^{15} > > (2^{16} \div 2^{16}) - 1$	$2^{16} \div 2^{16}$	0	0	10000000000000001	15	1	16
17	0	0	0	0	100000000000000001	16	1	17
X : 不必要								

図39

【国際調査報告】

INTERNATIONAL SEARCH REPORT

Int. application No.
PCT/US94/02088

A. CLASSIFICATION OF SUBJECT MATTER IPC(5) : Please See Extra Sheet. US CL : Please See Extra Sheet. According to International Patent Classification (IPC) or to both national classification and IPC		
B. FIELDS SEARCHED Minimum documentation searched (classification system followed by classification symbols) U.S. : 373/122; 364/900; 341/50, 51, 55, 60, 64, 65, 67, 79, 87, 90, 95, 106, 107; 382/40, 56 Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched Electronic data base consulted during the international search (name of data base and, where practicable, search terms used) Please See Extra Sheet.		
C. DOCUMENTS CONSIDERED TO BE RELEVANT		
Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US, A, 4,881,075 (WENG et al) 14 November 1989.	1-38
A	US, A, 4,955,066 (NOTENBOOM) 04 September 1990.	1-38
A	US, A, 4,782,325 (JEPPSSON et al) 01 November 1988.	1-38
A	US, A, 4,796,003 (BENTLEY et al) 03 January 1989.	1-38
A	US, A, 4,672,539 (GOERTZEL) 09 June 1987.	1-38
A	US, A, 4,597,057 (SNOW) 24 June 1986.	1-38
A	US, A, 4,545,032 (MAK) 01 October 1985.	1-38
<input checked="" type="checkbox"/> Further documents are listed in the continuation of Box C. <input type="checkbox"/> See patent family annex.		
* Special categories of cited documents: "A" document defining the general state of the art which is not considered to be part of particular relevance "E" earlier document published on or after the international filing date "L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified) "O" document referring to an oral disclosure, use, exhibition or other means "P" document published prior to the international filing date but later than the priority date claimed "T" later documents published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention "X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone "Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art "Z" document member of the same patent family		
Date of the actual completion of the international search 06 JUNE 1994		Date of mailing of the international search report JUL 11 1994
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230		Authorized officer  EDWARD L. COLES, SR. Telephone No. (703) 305-4712

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US94/02088

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT		
Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US, A, 4,491,934 (HEINZ) 01 January 1985.	1-38
A	US, A, 3,694,813 (LOH et al) 26 September 1972.	1-38

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US94/02088

A. CLASSIFICATION OF SUBJECT MATTER:
IPC (5):

H04B 1/66; G06F 1/00; H03M 7/00, 7/34, 7/38, 5/00, 7/00, 7/40; G06K 9/36, 9/46

A. CLASSIFICATION OF SUBJECT MATTER:
US CL :

375/122; 364/900; 341/50, 51, 55, 60, 64, 65, 67, 79, 95, 106, 107; 382/56

B. FIELDS SEARCHED

Electronic data bases consulted (Name of data base and where practicable terms used):

APS

search terms: data compression, nibble encryption, nibble encoding, concatenating, parsing,
direct bit manipulation, redundant data, packing, distribution compression